UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE MATEMÁTICA

ISCTE
INSTITUTO UNIVERSITÁRIO DE LISBOA
DEPARTAMENTO DE FINANÇAS

Modelo de *Financial Trading* através de Redes Neuronais Convolucionais

Carolina do Carmo Godinho

Mestrado em Matemática Financeira

Dissertação orientada por:
Diana A. Mendes

2020

Agradecimentos

Num ano particularmente diferente, que nos ensinou que não são apenas os Mercados Financeiros difíceis de prever e que será também marcado pela conclusão de uma etapa importante e que me empurrou para fora da minha zona de conforto. Este sentimento flutuante, doce e pleno só foi possível pelas pessoas que me acompanharam.

Aos meus avós, em especial à mulher mais batalhadora e bonita que conheço, a minha avó e que me deu tudo isto.

Aos meus pais e ao meu irmão, que sempre me deixaram voar e me ouviram ou aturaram quando as aterragens foram atribuladas.

Aos meus tios e primas que sempre me insuflaram de mimos.

Aos meus amigos de longa data e àqueles que fiz pelo caminho e que me deram, às vezes sem saber e involuntariamente a confiança desaparecida ou um copo de vinho ou cerveja no final do dia.

Aos professores que me fizeram gostar de Matemática, aguçaram a minha curiosidade e plantaram sempre muitos pontos de interrogação na minha cabeça. Em especial à professora Diana, pela paciência de quando lhe enchia a caixa de correio e pelo seu trabalho excepcional e empenho que coloca em todo o que faz.

Isto sabe tão bem, mas sabe sempre a pouco.

Resumo

Actualmente a Inteligência Artificial está presente um pouco por todas as áreas, nomeadamente nos Mercados Financeiros. No entanto, neste trabalho explora-se uma metodologia que foge à regra do que tem vindo a ser desenvolvido. Através de imagens de gráficos de barras que representam o preço de fecho de ações (*Dow Jones 30*) e com a utilização de uma Rede Neuronal Convolucional (CNN) tentou criar-se um novo modelo/algoritmo de negociação financeira.

Foi considerado o intervalo temporal de 1987-2020 para gerar as imagens, cada uma com uma janela temporal de 30 dias e estas foram o *input* para treinar o algoritmo da CNN com o objetivo final de determinar pontos de *Buy*, *Sell* ou *Hold*.

Os resultados indicaram que, apesar de este ser um estudo preliminar e possivelmente umas das primeiras tentativas de uma abordagem não convencional, o algoritmo conseguiu responder de forma favorável à identificação de pontos de *Trading*. De notar que devido ao estado embrionário deste estudo, há muita margem para que os resultados e conclusões sejam melhoradas, tal como outros ajustes que possam combinar diferentes estratégias ou algoritmos.

Palavras-chave: Algoritmo de *Trading*, *Deep Learning*, Matemática Financeira, Predição Financeira, Redes Neuronais Convolucionais.

Abstract

Nowadays, Artificial Intelligence is present in all areas, namely in Financial Markets. However, this work explores a methodology that is an exception to the rule of what has been developed. Through images of bar chart that represent the closing price of stocks (Dow Jones 30) and using a Convolutional Neuronal Network (CNN), a new financial trading model / algorithm was created.

Was considered the time interval from 1987 to 2020 to generate the images, each with a time window of 30 days and these were the input to train the CNN algorithm with the ultimate goal of determining the Buy, Sell or Hold points.

The results indicate that, although this was a preliminary study and probably one of the first attempts at an unconventional approach, the algorithm was able to respond favorably to the identification of Trading points. It should be noted that due to the embryonic state of this study, there is an opportunity for the results and conclusions to be improved, as well as other adjustments that might be combined in different strategies or algorithms.

Keywords: Algorithmic Trading, Convolutional Neural Networks, Deep Learning, Financial forecasting, Financial Mathematics.

Lista de Abreviaturas

Capítulo 1 e 2

AdaGrad *Adaptive Gradient Algorithm*

AI *Artificial Inteligence*

ANN *Artificial Neural Network*

BaH *Buy and Hold*

CNN *Convolutional Neural Network*

DL *Deep Learning*

DNN *Deep Neural Network*

ETFs *Exchange Trade Funds*

EMA *Exponential Moving Average*

GA *Genetic Algorithm*

HMA *Hull Moving Average*

LSTM *Long Short-Term Memory*

MACD *Moving Average Convergence and Divergence*

ML *Machine Learning*

MLP *Multilayer Perceptron*

MNIST *Modified National Institute of Standards and Technology*

ReLU *Rectified Linear Unit*

RNN *Recurrent Neural Network*

RMS *Root Mean Square*

RMSProp *Root Mean Square Propagation*

ROC *Receiver Operator Characteristic Curve*

RSI *Relative Strength Index*;

SAR *Stop and Reverse*

SGD *Stochastic Gradient Descendent*

SVM *Support Vector Machine*

TEMA *Triple Exponential Moving Average*

WMA *Weighted Moving Average*

Capítulo 3

AnT *Annualized number of Transactions*

ApT *Average percent profit per Transaction*

AR *Annualized Return*

IdleR *Idle Ratio*

L *Average transaction Length*

MAE Erro Absoluto Médio

MaxC *Maximum Capital*

MDD *Average Percentage of Maximum Drawdown*

MinC *Minimum Capital*

MlT *Maximum loss percentage in Transaction*

MpT *Maximum profit percentage in Transaction*

MSE Erro Quadrático Médio

PoS *Percentage of Success*

RoMaD *Return Over Maximum Drawdown*

Índice

Índice de Figuras

Índice de Tabelas

Introdução

O ser humano procura estar um passo à frente daquilo que faz, não se contentando com aquilo que já sabe, procurando formas de se reinventar ou melhorar e de saber mais.

A Matemática e os Mercados Financeiros, aliados às Ciências da Computação são exemplos dessa aprendizagem constante e profunda.

Nas últimas décadas várias abordagens foram propostas em relação à predição nos Mercados Financeiros de forma a dispor de sistemas de suporte à decisão do ser humano ou do investidor, neste caso em particular.

Modelar a complexidade, a não linearidade e não-estacionaridade, tal como os ruídos do mundo real através de dados, sendo estes (séries temporais) um tipo de dados bastante particular torna-se portanto um desafio difícil de superar. Essa complexidade provem de várias teorias demonstradas ao longo dos anos e que argumentam, por exemplo, que os preços não podem ser preditos, pois o Mercado Financeiro é *informationally eficiente* [Fama, 1964], ou seja, o preço das ações é reflexo de toda a informação e o seu movimento é uma resposta a notícias ou eventos.

A aplicação de métodos de *Deep Learning* (DL) nos Mercados Financeiros é ainda uma área por explorar tal como a predição dos mesmos. O desafio principal passa por encontrar arquiteturas de DL que consigam adaptar-se à imprevisibilidade dos dados e detetar padrões de *trading* de forma a diminuir o risco nestas estratégias, tal como aumentar a eficácia e rapidez nas decisões.

Na actualidade, *Fuzzy systems*, *Suported Vector Machine* (SVM), *Artificial Neural Network* (ANN), *Evolutionary Algorithms*, mecanismos híbridos, *Ensemble Methods* são dos métodos mais usados na predição em Mercados Financeiros. O uso destes algoritmos encontra-se, por exemplo, nos Mercados de Ações e nos de Forex.

No universo de redes neuronais artificiais (*Artificial Neural Networks* - ANN) e de aprendizagem profunda (*Deep Neural Networks* - DNN) – tipo de rede neuronal altamente parametrizada composta por múltiplas camadas, muitas delas escondidas, o que lhe confere o poder de conseguir capturar características mais importantes de uma forma mais "abstrata". A sua popularidade tornou-se evidente ao ter melhorado de forma significativa temas como reconhecimento de voz e objectos e muitos outros domínios. É de destacar redes como *Recurrent Neural Network* (RNN), *Long Short Term Memory* (LSTM), *Convolutional Neural Network* (CNN) ou *Multilayer Perceptron* (MLP).

Neste trabalho pretende-se analisar os resultados da utilização de um tipo de rede, pouco usada em previsão nos Mercados Financeiros, mas com enorme sucesso na classificação de imagens. Uso de uma Rede Neuronal Convolucional (CNN) com gráficos de barras ao invés de dados de séries temporais. Convertendo informação de preços, em gráficos de barras representativos desses preços e dos seus movimentos. Cada imagem corresponde ao preço da ação durante 30 dias, resultando numa imagem de 30x30 pixéis. Usando uma CNN, o modelo é treinado de forma a avaliar se para um determinado preço estamos perante uma situação de compra, venda ou se apenas devemos manter (*Hold*) a posição em questão. Perante os resultados obtidos estes serão comparados com uma estratégia de *Buy&Hold* (BaH), comumente usada nos Mercados Financeiros.

No estudo original de Sezer & Ozbayoglu (2019) utilizaram-se os dados do *Dow Jones 30* de 1997 a 2017 com dois diferentes conjuntos de dados de teste (2007-2012 e 2012-2017). Porém para um contributo positivo a este artigo, nesta dissertação utilizou-se os dados do *Dow Jones 30* (Microsoft) de 1987 a 2020 e foi alterado o cálculo e rotulagem das imagens, tal como o código da rede neuronal. Para este trabalho foi utilizada linguagem de programação *Python.*

O trabalho desenvolvido tem uma estrutura que pretende representar as etapas principais de predição: Preparação dos dados, definição do algoritmo a usar, treino do algoritmo, predição e avaliação (lucro em mercado). Para obter uma visão contextualizada deste tema o Capítulo 1 aborda os temas mais teóricos (*Machine* e *Deep Leaning*), com um resumo da rede neuronal utilizada (CNN) e a revisão de literatura tem o intuito de contextualizar no tempo os trabalhos que impactam com o que foi abordado neste trabalho. De seguida, o Capítulo 2 relaciona o tema de *Machine Learning* e Modelos de *Trading*, nomeadamente os desenvolvimentos e abordagens mais relevantes até aos dias de hoje. Por fim, o Capítulo 3 apresenta a Análise Empírica que abrange a explicação da metodologia utilizada tal como os resultados obtidos.

A dificuldade de previsão ainda acaba por limitar o que é feito nesta área, pois muito do que está implementado no dia a dia dos Mercados Financeiros do mundo inteiro não está em artigos acessíveis a toda a comunidade científica. Porém esta dificuldade aguça o engenho e aumenta o incentivo para continuar a explorar uma área com múltiplos caminhos ou redes para percorrer.

Referir também que esta é uma das primeiras pesquisas que aborda a integração de um algoritmo de DL de classificação de imagens e um sistema de decisão de compra-venda de posições em mercados acionistas e em Portugal é um tema ainda por explorar, não havendo pelo menos trabalhos finais sobre o mesmo.

1. *Machine Learning, Deep Learning*, Redes Neuronais & *Convolutional Neural Network*

1.1 *Machine Learning & Deep Learning*

Hoje a Inteligência Artificial é um tema presente nas nossas vidas e que possui as mais variadas aplicações práticas, como por exemplo, reconhecimento de imagens e voz, está presente em quase todas as áreas, desde os Mercados Financeiros, Banca, Seguradoras, Marketing ou Medicina. E serve para solucionar problemas que continuariam difíceis de resolver sem ajuda computacional.

A solução encontrada para esses problemas tem como base a aprendizagem computacional a partir da experiência e da forma como os dados se relacionam a partir de conceitos simples.

A experiência leva a que seja necessária a intervenção activa do Humano. Dessa necessidade e da dificuldade do passado de sistemas/programas baseados em *hard coded* surgiu o conceito de *Machine Learning* – capacidade de extrair padrões de dados não tratados/trabalhados para obter conhecimento.

A introdução de *Machine Learning* direccionou a Ciência de Computação (*Data Science*) a ser capaz de resolver e tomar decisões em assuntos do mundo real e de certa forma subjetivos.

Um exemplo simplificado de ML, *Support Vector Machine* (SVM), este modelo assemelha-se à regressão logística, visto que também é definido por uma função linear. Porém ao contrário da função logística não retorna probabilidades, apenas retorna a identidade de uma classe (classe positiva está presente quando a função é positiva e a classe é negativa quando a função é negativa). Este modelo foi utilizado, por exemplo, para a predição do preço do crude (Bao *et al.* 2011).

Uma solução encontrada também em *Machine Learning* para saber que *features* (características) devem ser extraídas é um tipo de abordagem conhecida como *representation learning*. O problema desta abordagem provém de existirem inúmeras variações e factores que podem influenciar cada "piece" de dados, ou seja, continua a existir uma dificuldade em extrair dados e as suas características mais abstratas e sofisticadas (*high level features*).

O *Deep Learning* surge então desse problema central do *representation learning*, introduzindo representações que são expressas em funções de outras mais simples.

O DL corresponde a um modo de aprendizagem que segue uma hierarquia de conceitos, permitindo que o "algoritmo" aprenda conceitos mais complexos, construindo-os através de outros mais simples.

Se tivéssemos uma representação gráfica que mostrasse como esses conceitos são construídos em cima uns dos outros, seria um gráfico "profundo" ("*deep*") com várias camadas (*layers*). Portanto, DL é um tipo de abordagem à Inteligência Artificial.

Um exemplo de um modelo de DL é a *Feedforward Deep Network*, tendo também o nome de *Multilayer percepton* (MLP), que é apenas uma função matemática que mapea um certo conjunto de vários *inputs* para um determinado conjunto de valores de *output*. E cada aplicação de uma função diferente faculta uma nova representação do *input*.

O termo ou facto de ser "*Deep*" pode ser reflexo do tamanho da representação gráfica a percorrer até se obter uma resposta final. Como também se pode dever à complexidade a nível computacional, tal como a forma como os conceitos estão relacionados.

Em resumo, DL é uma abordagem de AI e um tipo de *Machine Learning* (uma técnica que permite a um sistema de computadores melhorar com a experiência e com dados). O DL alcança grande poder e flexibilidade ao representar o mundo como uma hierarquia de conceitos, com cada conceito definido em relação a conceitos mais simples e representações mais abstratas determinadas em termos menos abstratos.

O DL tem uma longa história e passou até por várias designações. Tornou-se mais útil quando os dados se tornaram mais acessíveis e tem melhorado ao longo dos anos. Inicialmente conhecido por *Cybernetics* (1940-1960), *Connectionism* (1980-1990) e a partir de 2006 surgiu com o nome de *Deep Learning*.

Hoje reconhece-se que alguns dos mais recentes algoritmos de aprendizagem têm a intenção de serem modelos computacionais de aprendizagem biológica – modelos fundamentados na aprendizagem que acontece ou deveria acontecer no cérebro. Desta abordagem surge o nome de *Artificial Neural Networks* (ANN), que acaba por corresponder a um sistema computacional/engenharia inspirado no cérebro.

A era do *Big Data* tornou o ML e a AI mais fáceis de serem colocadas em prática no mundo real. Para este sucesso também contribuíram o aumento do tamanho dos modelos e o aumento da capacidade computacional.

Multi-Layer Neural Network (MLNP), *Recurrent Neural Network* (RNN), *Long -Short Term Memory Neural Network* (LSTM) e *Convolutional Neural Network* (CNN) são as principais e mais utilizadas redes neuronais de DL usadas nos dias de hoje. Outra conquista do DL foi o *Reinforcement Learning*, que foi melhorando através deste tipo de aprendizagem.

Para entender o DL é necessário entender os princípios básicos de ML. Os algoritmos de aprendizagem dividem-se em três grandes grupos:

Supervised **-** Os algoritmos de ML supervisionados contêm um conjunto de dados em que cada exemplo também está associado um rótulo (*label*).

Existem vários modelos/algoritmos de aprendizagem supervisionada, como por exemplo, *ANN, SVM, Bayesian networks, Decision trees, Ensembles of classifiers* ou *Instace-based learning models.*

Unsupervised **-** Os algoritmos de ML não-supervisionados experimentam um conjunto de dados contendo muitos recursos e aprendem propriedades (*features*) úteis da estrutura desse conjunto de dados. *Self-organizing maps, Associated rule learning, Clustering techniques*, são alguns dos exemplos deste tipo de algoritmo.

No contexto de DL, geralmente queremos aprender a distribuição de probabilidade gerada pelo conjunto de dados fornecidos seja de forma explícita como na estimativa da densidade de probabilidade ou implicitamente, para tarefas como síntese ou *denoising*. Resumidamente, a aprendizagem não-supervisionada envolve a observação de vários exemplos de um vetor aleatório x e a tentativa de aprender implícita ou explicitamente a distribuição de probabilidade p(x) ou algumas propriedades

interessantes dessa distribuição; enquanto a aprendizagem supervisionada envolve a observação de vários exemplos de um vetor aleatório x e um valor ou vetor associado y, aprendendo a prever y a partir de x, geralmente estimando p (y | x). O termo predito resulta de uma instrução ou de um guia de aprendizagem, enquanto nos não-supervisionados o algoritmo deve aprender a entender os dados sem este guia.

Reinforcement Learning - Algoritmos de ML que não têm apenas um conjunto de dados fixo e interagem com um ambiente, para que haja um ciclo de *feedback* entre o sistema de aprendizagem e suas experiências. *Value-based, Policy-based, Model (actor)-based* representam alguns exemplos deste tipo de aprendizagem.

1.2 Redes Neuronais

"The design of an ANN is more of an art than a science" (Zhang *et al*., 1998)

Para melhor entender o estado da arte, o funcionamento do DL e em particular da rede neuronal utilizada nesta dissertação (*Convolutional Neuronal Network*) é necessário explicar de uma forma sucinta a base destes algoritmos.

As redes neuronais estão organizadas em grupos de unidades denominadas por camadas (*layers*). Estas camadas estão sempre de certa forma encadeadas, dado que o resultado (*output*) de uma camada é sempre a base de trabalho (*input)* da camada seguinte.

Todas as camadas são constituídas por neurónios, a unidade base do algoritmo.

E entre a camada de *input* e *output*, encontram-se as camadas escondidas (*hidden layers*) que, de certa forma quantificam o quão complexa é a rede, pois quanto maior for o seu número mais complexa é.

$$\mathrm{Y} = \sum \text{peso} \times input + bias$$

Cada neurónio (Y) é a soma ponderada de todos os *inputs* que recebe à qual se adiciona a "bias" (tendência) que funciona como uma constante que leva a mudanças na função de activação das camadas escondidas e que consequentemente leva a que o *ouput* esteja mais próximo do valor real.

"Como é que o algoritmo aprende?" ou **"Onde se esconde a Matemática nestas camadas?"** foram algumas das perguntas que não "calaram" a mente de quem desenvolveu ou estudou estes temas.

Praticamente todos os algoritmos de DL são compostos por: Dados, a *Cost/Loss function*, que não é nada mais do que uma função objetivo, e um algoritmo de optimização.

Os dados funcionam como uma coleção de exemplos e a maneira mais fácil de os representar, visto que estamos perante milhares ou milhões de exemplos é através de uma matriz.

Durante a aprendizagem, estes dados que estão categorizados (*supervised*) são "carregados" no algoritmo na tal camada de *input* e esta produz um resultado na camada final. Esse resultado final corresponde à mesma categoria do *input* ou é o resultado expectável caso a aprendizagem esteja a acontecer corretamente. Num exemplo prático, se o *input* fosse uma imagem de uma pessoa, o algoritmo teria classificado a mesma imagem como sendo uma pessoa.

Em relação à computação da *loss/cost function*, ***Negative log-likelihood***, é a função frequentemente usada, visto que ao minimiza-la leva à estimação de máxima verosimilhança[1].

Em seguida, o algoritmo modifica internamente os seus parâmetros de forma a reduzir o erro. Estes parâmetros já referidos acima são os pesos. E tal como os dados, estes são também representados matricialmente. De forma a ajustar a matriz com os pesos, o que se pretende é simplesmente encontrar um mínimo. Visto que não estamos apenas numa dimensão e são inúmeros os parâmetros que pretendemos minimizar, temos de encontrar uma direção que nos leve à diminuição do valor da função, ou seja, necessitamos de calcular o gradiente. Neste caso como se pretende a direção para a qual a nossa função diminui queremos o gradiente negativo. O gradiente indica o impacto da alteração dos pesos no aumento ou diminuição do erro. O vector dos pesos é então ajustado na direção oposta ao vector gradiente.

Esta é uma explicação simples de um algoritmo de optimização cujo objetivo é encontrar um mínimo de uma função (*cost function*) e, por conseguinte, este é definido de forma a resolver onde/em que pontos o gradiente da função é zero.

O *Stochastic Gradient Descent* (SGD), que representa apenas uma estimação do gradiente e as suas inúmeras adaptações é dos algoritmos de optimização mais usados em ML e em particular em DL.

O gradiente dessa função a minimizar é calculado por *Back-propagation (*Retropropagação) no caso de uma CNN.

O algoritmo de Retropropagação (Rumelhart *et al*., 1986), geralmente chamado de *backprop*, permite que as informações da função custo se propaguem da camada final para a camada inicial da rede para calcular o gradiente.

Determinar uma expressão analítica para o gradiente é simples, mas avaliar numericamente essa expressão pode ser computacionalmente difícil ou pesado e é por isso que este método é tão usado, pois tem um procedimento simples e pouco exigente a nível computacional.

Os três esquemas seguintes explicam as partes importantes do cálculo do algoritmo e de como é propagado pela rede[2]:

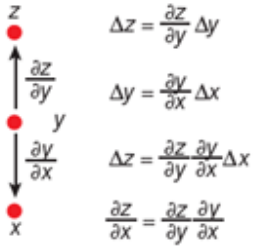

Figura 1: Regra da cadeia de derivadas[3]

Figura 1- Pequena variação Δx no x é transformada primeiro numa pequena variação Δy em y ao ser multiplicado por $\frac{\partial y}{\partial x}$ *(*derivada parcial).

Da mesma forma a variação Δy leva a uma variação Δz no z. Substituindo uma equação por outra resulta na regra da cadeia de derivadas – Δx transforma-se em Δz através da multiplicação de $\frac{\partial y}{\partial x}$ e $\frac{\partial z}{\partial y}$. No caso de x, y e z serem vetores as derivadas parciais são representadas numa Matriz Jacobiana. Esta operação descreve-se como obtendo dois pequenos efeitos (variar o x, provoca efeito no y e ao variar o y provoca efeito no z).

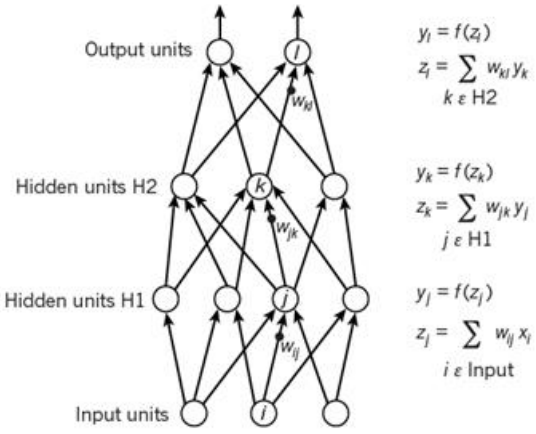

Figura 2: Esquema de propagação da Rede Neuronal entre camadas[3]

[1] Livro Goodfellow & Courville (2016) pode ser consultada a expressão de função custo (*loss function*).

[2]Consultar LeCun, Bengio & Hinton (2015) para mais pormenores

Figura 2- Na camada de input (i), o valor z presenta a soma ponderada dos dados de input (x_j) e os pesos que ligam a camada i à camada j (w_{ij}). Em seguida este valor é passado para uma função não linear (por uma questão de simplicidade o termo de "bias" é omitido). Este valor é depois passado à camada seguinte como a forma de $w_{jk}y_j$, que representa, o output da camada. Nas unidades das camadas escondidas, o valor interno da camada resulta da soma ponderada dos outputs da camada anterior com os pesos que a ligam à camada seguinte (k).

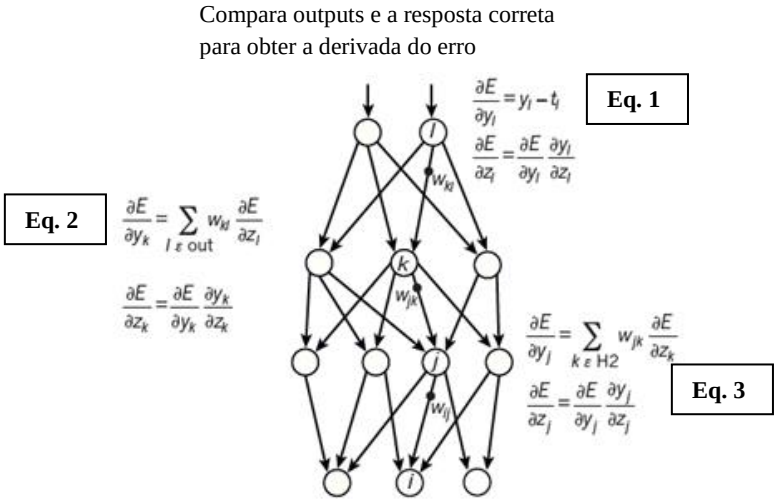

Figura 3: Esquema de Backpropagation[3]

Figura 3 – Sumariza as equações usadas no cálculo do gradiente, da passagem da camada final para as camadas anteriores até chegar à camada inicial.

O algoritmo inicia na camada de *output*, a derivada do erro em relação à função de *output* (y_l) é dada através da diferenciação da *cost function*. Por exemplo, isso teria como resultado diferença entre o valor de output(y_l) e o valor real/ *target value* (t_l) se a *cost function* fosse $0.5(y_l - t_l)^2$- **Equação 1**.

Em cada camada escondida, o cálculo das derivadas do erro (E) em relação ao output de cada unidade da respetiva camada (y_l) – **Equação 2**, é a soma ponderada da derivada do erro em relação ao total de inputs das unidades na camada a seguir (z_l) e os pesos que ligam essas duas camada (w_{kl}). Para converter a Equação 2 na Equação 3 (passar da derivada em função do output para ser em função do input-Regra da cadeia), basta multiplicar pelo gradiente de $f(z_j) = y_j$. Uma vez que $\frac{\partial E}{\partial z_k}$ é conhecida (Equação 2), a derivada do erro para o peso w_{jk} na conexão de j com a camada abaixo é dado por $f(z_j)\frac{\partial E}{\partial z_k}$ **(Equação 3).**

[3]Imagens retiradas de LeCun, Bengio & Hinton (2015)

O cálculo do gradiente da função objetivo em função dos pesos de múltiplas camadas, na prática é apenas a regra da cadeia das derivadas. A chave é que a derivada ou o gradiente da função em relação ao *input* dessa camada pode ser calculada de trás para a frente com respeito ao *output* da mesma ou como o *input* da camada a seguir. O algoritmo de *Back-propagation* pode ser aplicado repetidamente de forma a propagar o gradiente por toda a rede, começando no *output* (valor predito) até à camada inicial (*input* vindo dos dados de treino). Depois destes gradientes serem calculados é direto o cálculo dos gradientes em função dos pesos de cada módulo ou camada, ou seja, através da derivada de um único *output* consegue-se obter as derivadas em função dos parâmetros de *input*.[4]

Para ir de uma camada para outra, um conjunto de unidades calculam uma soma ponderada dos seus *inputs* da camada anterior e passa o resultado através de uma função de ativação. É através desta função que se decide qual a camada escondida que deve ser ativada. Actualmente a função não-linear mais usada é ReLU (*Rectified Linear Unit*) definida por: $f(z) = max(z, 0)$. Outras funções de ativação usadas nos modelos de redes neuronais são, por exemplo, a tangente hiperbólica $\tanh(z)$ e a logística $\frac{1}{1+e^{-z}}$, porém ReLU aprende de forma mais rápida em redes com muitas camadas.

No cerne do algoritmo na última camada o que existe é a probabilidade, por exemplo, de uma imagem representar uma pessoa num conjunto de imagens que foram apresentadas. Para essa conversão, geralmente utiliza-se a ***Softmax function*.**

Softmax é uma forma de regressão logística que normaliza o valor inicial num determinado vector de valores que seguem uma distribuição de probabilidade em que soma total é 1. Os valores de *output* estão compreendidos entre os valores 0 e 1.Esta função transforma uma camada com *output* numérico, neste caso a última camada, numa probabilidade. Essa probabilidade resulta por utilizar como expoente um valor do vetor dessa camada e em seguida normalizá-lo pela soma de todos os exponentes dos valores que constituem o vector da camada de *output*. *Softmax* é frequentemente adicionada na última camada de uma CNN.

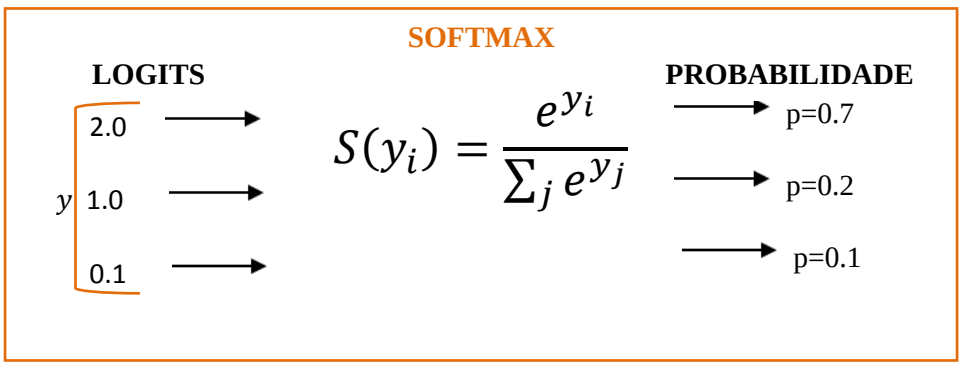

Figura 4: Softmax[5]

[4] Explicação de uma forma simples do algoritmo de *Back- propagation* em http://colah.github.io/posts/2015-08-Backprop/

Em cada iteração do algoritmo é usado um *batch* de exemplos provenientes do conjunto total dos dados de treino e este valor é fixo logo à partida. É através de cada *batch* que vão sendo feitas pequenas minimizações até chegar a um mínimo local. O que implica que o processo ao ser iterativo seja computacionalmente mais rápido do que encontrar um mínimo global que implicaria um processo mais demorado. E, por fim, o número de vezes que os dados de treino são processados tem o nome de *epochs*. Uma época (*epoch*) corresponde a um ciclo completo sobre todo o conjunto de treino. Geralmente utiliza-se várias épocas para treinar a rede, que podem ser entre 1 e 1000, por exemplo.

Depois do treino, a *performance* do modelo é avaliada através de dados de testes, dados diferentes dos de treino, de forma a testar a habilidade de generalização do modelo. É nesta fase que se observa a capacidade do algoritmo prever corretamente dados novos.

1.3 *Convolutional Neural Network*

As *Convolutional Neural Network* (CNNs) são talvez o exemplo com maior sucesso de como a Biologia inspirou a Inteligência Artificial.

A história da CNN começou com experiências neurocientíficas (Hubel & Wiesel, *et al*. 1968) nas quais observaram como os neurónios do cérebro de um gato respondiam a imagens projetadas em locais específicos de um ecrã. E descobriram que os neurónios no sistema de visão respondem de forma mais forte a muitos padrões específicos de luz, porém respondem pior ao resto de outros padrões.

De uma forma simplista, V1 (primeiro córtex visual) é a primeira zona do cérebro que começa a processar as imagens. E cada camada de uma CNN foi desenhada para capturar três propriedades do V1:

V1 é organizado num mapa espacial, da mesma forma que a CNN tem as suas características definidas em termos de mapas que podem ter várias dimensões (*feature maps*).

V1 contem inúmeras células simples que são ativadas ao processarem campos específicos numa imagem, tal como as unidades de deteção de rede.

E, por último, V1 também contem células complexas que são invariantes a pequenas mudanças, tal como as unidades de *pooling* presentes na rede.

CNN (LeCun, *et al*. 1989) é um tipo de rede neuronal que processa dados que estão dispostos em forma matricial, como imagens 1D, 2D ou 3D. É na operação de convolução que está a sua particularidade, uma operação matemática usada em pelo menos uma das camadas da rede em vez de uma multiplicação de matrizes.

[5]Imagem original retirada do *website* https://medium.com/data-science-bootcamp/understand-the-softmax-function-in-minutes-f3a59641e86d

A operação de convolução, de um modo mais geral possível, é uma operação matemática que "junta" duas funções para produzir uma terceira função que expressa uma alteração que ocorreu numa dessas funções iniciais. É uma ferramenta poderosa, que surge em várias vertentes da Matemática, que vai desde o estudo de equações diferenciais parciais até a teoria da probabilidade. Para este tipo de rede neuronal em específico torna a sua implementação mais fácil, tal como a sua formulação. Por fim elimina cálculos desnecessários e por conseguinte tornam esta rede mais rápida e eficiente.[6]

Esta operação pode ser expressa de uma forma discreta:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{+\infty} x(a)w(t-a)$$

No contexto de redes neuronais, o primeiro argumento x representa o *input*, o w é o Kernel. E o s, é referido como o *feature map* – nome dado à forma como as unidades das camadas convolucionais são organizadas. Esta primeira fórmula apenas representa a operação de convolução num eixo (1D, onde t representa a variável tempo).

Em ML o *input* é normalmente uma matriz multidimensional de dados e o Kernel é também uma matriz multidimensional, neste caso de parâmetros (pesos), que são adaptados pelo algoritmo de aprendizagem. Neste tipo de rede em específico, estas matrizes multidimensionais são muitas vezes referidas como *tensors*.

Tantos os valores de *input* como os valores do Kernel são introduzidos de forma explícita e separada, assumindo que estas funções são sempre zero, excepto num conjunto finito de pontos. Isto implica que em vez de termos uma soma infinita de elementos, temos uma soma sobre um número finito de elementos de uma matriz.

Para complementar a fórmula anterior e tendo em conta que este trabalho final tem como base imagens em duas dimensões, a operação de convolução é usada em mais do que um eixo. Pois ao usar-se como *input* uma imagem a duas dimensões, I, teremos também um Kernel com duas dimensões e temos como resultado a equação seguinte:

$$S(i,j) = (I * K)(i,j) = \sum_{m}\sum_{n} I(m,n)K(i-m, j-n)$$

[6]Para uma descrição mais pormenorizada e sintética acerca da operação de convolução para o entendimento de como funciona uma CNN consultar: https://colah.github.io/posts/2014-07-Understanding-Convolutions/

Enquanto numa rede comum a multiplicação entre a matriz de *input* e matriz de parâmetros (pesos) descreve a interação entre cada unidade de *input* e cada unidade de *output*, isso não acontece da mesma forma numa CNN, pois o Kernel é sempre uma matriz de menor dimensão do que a matriz de *input*. Isto leva a que, por exemplo, numa imagem com milhares de pixéis sejam detetados pormenores (cantos, por exemplo) de forma a que seja mais fácil identificar o que está representado naquela imagem. Como consequência ao apresentar menos parâmetros reduz a memória necessária do modelo e melhora a eficiência, pois para ter uma resposta são necessárias menos operações.

Outra característica importante deste tipo de rede é a partilha de parâmetros. Numa rede neuronal comum cada elemento da matriz de pesos é usado exatamente uma única vez quando é calculado o *output* de uma camada. Na CNN o peso de um elemento da matriz nessa camada é usado/incorporado nos outros valores dos pesos presentes no Kernel. Os dados em matrizes, como por exemplo em imagens, os grupos de valores locais estão altamente correlacionados, formando padrões locais distintos que são facilmente detetados. Isto implica que a aprendizagem seja feita num conjunto de parâmetros e não apenas num parâmetro em específico. E, por consequência, uma parte de uma imagem pode aparecer em qualquer lugar, daí a ideia de unidades em locais diferentes partilharem os mesmos pesos e detetarem o mesmo padrão em diferentes partes da matriz.

A arquitetura típica deste tipo de rede neuronal é estruturada por uma série de etapas. As primeiras etapas são compostas por dois tipos de camadas: Convolucionais e *Pooling*.

Durante o funcionamento da rede estas são as etapas a destacar:

-A camada executa várias operações de convolução em paralelo de forma a produzir um conjunto de ativações não lineares. Cada unidade é conectada a segmentos locais no *feature map* da camada anterior por um conjunto de pesos- *filter bank.*

-Cada ativação não linear, resultado dessa soma local (dos pesos) é executada por uma função de ativação não linear (ReLU referenciada anteriormente).

- Uso de uma *pooling function* que modifica o *output* para a camada a seguir.

A ***pooling function*** substitui o *output* da rede numa certa localização com um "resumo"/sumário estatístico dos *outputs* próximos. A operação de ***max pooling*** (Zhou *et al*. 1988) reporta o *output* máximo dentro de uma vizinhança rectangular.

Esta operação ajuda a tornar a representação aproximadamente invariante a pequenas translações do *input*.

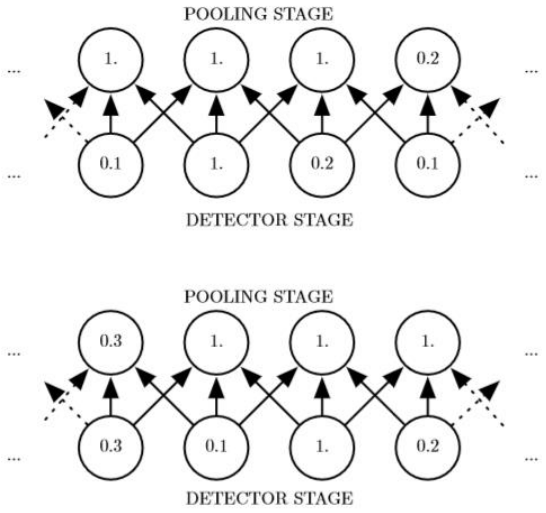

Figura 5: Esquema da fase de *Max Pooling*[7]

A **Figura 5** ilustra um exemplo do *ouput* de uma camada intermédia de convolução. A linha debaixo representa o *output* da função de não linearidade. A linha de cima representa o *output* do *max pooling*, com um intervalo de um pixel entre as regiões de *pooling* e uma região de *pooling* de 3 pixéis de largura.

O esquema debaixo representa a mesma rede, depois do *input* ter-se movimentado um pixel para a direita. Todos os valores da linha de baixo mudaram, porém apenas um valor da linha de cima mudou (50%), pois as unidades de *max pooling* são sensíveis apenas a valores máximos da vizinhança e não em relação à sua exata localização.

O uso de uma função de *pooling* resume os valores de *input* de uma determinada vizinhança, como se fundisse dados semelhantes num único. E ao levar a uma menor necessidade de unidades de *pooling* do que de "deteção", a eficiência da rede será maior, visto que a próxima camada terá menos inputs para processar.

Porém para que tudo isto funcione e sendo um dos pilares do algoritmo é necessário ter também um algoritmo de optimização. Como referido no ponto **1.2 Redes Neuronais** este processo de optimização através do SGD é um processo iterativo em que cada parâmetro x é actualizado pelo gradiente g dos parâmetros em cada i iteração e uma taxa de aprendizagem μ.

$$\Delta x_i = -\mu g_i$$

Esta taxa controla a magnitude do passo em que se dirige o gradiente. Quando a taxa de aprendizagem é demasiado elevada, leva a que função objetivo divirja; em contrapartida, se for demasiado pequena resulta num processo de aprendizagem demasiado lento.

[7]Imagem original retirada do livro Goodfellow & Courville (2016), Deep Learning.

Não existe um consenso de qual o melhor algoritmo a utilizar, pois esta escolha irá depender muito dos próprios parâmetros da rede e do seu algoritmo. Todavia os algoritmos de optimização mais utilizados e populares (RMSProp, AdaDelta e Adam) têm todos algo em comum, todos apresentam uma taxa de aprendizagem que é adaptada (*adaptive learning rate*). Será destacado apenas o AdaDelta visto que foi o utilizado no meu trabalho.

AdaDelta (Matthew *et al.* 2012) [8] é um método de optimização baseado numa taxa de aprendizagem que é adaptada por dimensão de forma a resolver os pontos fracos encontrados do método AdaGrad[9]. Nomeadamente o declínio contínuo da taxa de aprendizagem ao longo do treino da rede tal como a necessidade de ser introduzida uma taxa de aprendizagem inicial.

Este método adapta a taxa de aprendizagem com base numa janela móvel *w* dos grandientes, em vez de acumular a soma quadrada de todos os gradientes anteriormente calculados. Isto leva a que esta taxa continue a ser actualizada e, por consequência, os parâmetros, mesmo após muitas iterações/ actualizações da mesma.

A expressão que implementa a acumulação dos gradientes passados resulta da *Exponential Moving Average* (EMA) do quadrado do gradiente. ρ representa a constante de declínio (*decay*).

$$E[g^2]_t = \rho\, E[g^2]_{t-1} + (1-\rho) {g^2}_t$$

A expressão acima torna-se a média quadrática/quadrado médio da raiz (*RMS*) do gradiente quadrado anterior em relação à iteração t**.** $\boldsymbol{\varepsilon}$ é uma constante que é adicionada de forma a garantir que na primeira iteração se tenha, $\Delta x_0 = 0$, como também garante que continue a atualização dos parâmetros, mesmo que a atualização anterior tenho sido pequena.

$$RMS[g]_t = \sqrt{E[g^2]_t + \varepsilon}$$

A atualização dos parâmetros é dada então pela seguinte expressão:

$$\Delta x_t = -\frac{RMS[\Delta x]_{t-1}}{RMS[\Delta g]_t} g_t$$

O sinal negativo leva a que o método tenha o mesmo comportamento descendente que o SGD. O numerador é um termo de aceleração, que acumula os gradientes anteriores numa determinada janela de tempo. E o denominador que é idêntico ao usado no AdaGrad, ou seja, contém a informação da raiz do gradiente por dimensão o que ajuda a que o algoritmo avance em cada dimensão, mas no AdaDelta sempre em relação à janela temporal de forma a garantir que exista esse avanço mesmo até ao final do treino.

A CNN foi também uma das primeiras *Deep Networks* treinadas com *back-propagation*. Ainda não é claro o motivo pelo qual as CNNs foram das poucas redes que usaram *backprog* que tiveram sucesso, porém talvez seja porque estas são computacionalmente mais eficientes do que uma tradicional rede neuronal totalmente conectada.

[8] Para além da apresentação e explicação deste algoritmo, apresenta várias experiências utilizando o AdaDelta. Em particular os resultados são bastante satisfatórios quando este é usado numa CNN (MNIST e reconhecimento de voz). Os resultados obtidos podem ser consultados em maior detalhe no artigo de Matthew (2012).

[9]Explicado em maior pormenor no artigo de Matthew (2012)

1.4 Revisão de Literatura

Hinton *et al.* (1988) introduziram o uso de *backpropagation* para treinar redes neurais de atraso temporal (TDNNs). Para minimizar a terminologia contemporânea, as TDNNs são redes convolucionais unidimensionais aplicadas a séries temporais. A *backpropagation* aplicada a esses modelos não foi inspirada por nenhuma observação neurocientífica e é considerada por alguns como biologicamente pouco plausível. Após o sucesso no treino de TDNNs, LeCun *et al*. (1989) desenvolveram a moderna Rede Convolucional aplicando o mesmo algoritmo de treino à convolução 2-D aplicada às imagens.

Somi (2011) apresenta uma pesquisa acerca da aplicação de redes neuronais artificiais na predição de Mercados Financeiros.

Esta pesquisa defende o uso de redes neuronais face aos modelos tradicionais de predição. Apesar de ser possível modelar essa não linearidade por técnicas estatísticas presentes nos modelos tradicionais, esta característica tem de ser especificada antes da estimação dos parâmetros para a construção do modelo e ainda assim existe uma maior probabilidade de uma predição menos eficaz se compararmos com as redes neuronais. O facto de ANN serem modelos orientados pelos dados, através deste consegue-se descobrir a relação não linear entre os dados de entrada e os de saída e de adaptar internamente os seus parâmetros sem existir um pressuposto à partida. E tendo em conta que muitos destes dados são não-estacionários a rede consegue alterar o seu algoritmo em tempo real, enquanto noutros modelos isso não é possível.

Visto que este trabalho tem como fonte de dados os Mercados Financeiros é também importante ter-se uma ideia do que são os Mercados Financeiros, como são constituídos, para que servem ou que teorias existem por detrás da sua existência.

De realçar a sua importância como uma fonte das empresas ganharem dinheiro. A história tem vindo a mostrar como o preço de ações e outros ativos financeiros é uma parte importante da atividade económica e tem um papel de influenciar ou indicar o clima social do presente.

As teorias dos Mercados Financeiros e a predição dos seus movimentos também têm sido estudadas incansavelmente e são uma ajuda na construção de técnicas de predição.

A análise técnica e fundamental emergiu de duas abordagens: Hipótese de Mercados Eficientes e Teoria de Caminho Aleatório (*Theory of Random Walk*).

A primeira foi introduzida por Fama (1965) e defende que nenhuma forma de informação pode ser usada para gerar lucros extraordinários através dos Mercados Financeiros, pois os preços refletem completamente toda a informação disponível. Ou seja, qualquer informação que surja é absorvida de forma rápida e eficiente pelo preço e por isso o preço futuro de um ativo financeiro é impossível de prever tendo em conta o passado do mesmo.

A segunda abordagem defende que o preço não depende do passado, portanto os preços apresentam caminhos ou padrões aleatórios e que não podem ser explorados pois também não apresentam qualquer tendência. Porém as melhorias computacionais podem agora explorar esses padrões e os preços subjacentes, também devido à quantidade de dados que existem. Mas em contrapartida a *Theory of*

Random Walk afirma que apesar de essa informação estar disponível e poder ser utilizada, a predição continua a não ser eficaz.

E por último, esta pesquisa apresenta alguns dos trabalhos feitos nesta área tal como os modelos utilizados - MLP, RNN, Algoritmo Genético - e o uso de diferentes parâmetros, análise fundamental, médias móveis e valor de fecho de índices para predição nos mercados.

A utilização de uma CNN normalmente utilizada para a classificação de imagens é descrita por Krizhevsky *et al* (2012). Esta rede neuronal apresenta 5 camadas convolucionias, uma delas de *max pooling* e outras três totalmente conectadas e uma função de *softmax* na camada final.

Em todas as respostas das camadas é utilizada a função não linear de ativação, ReLu. Os autores indicam que apesar de se poder utilizar como funções de ativação a tangente hiperbólica ou uma função sigmoide, com ReLu a taxa de erro da aprendizagem diminui de forma muita acentuada logo após algumas iterações[10].A tangente hiperbólica (*tanh*) "satura" quando os valores de *input* são muito elevados ou negativos, ou seja, o declive da função nestes pontos fica próximo de zero o que diminui o valor do gradiente. No caso de ReLu o declive é sempre diferente de zero quando o *input* é positivo; quando este é negativo o declive é zero, o que não é um problema porque a maior parte dos valores de input são positivos.

Outro pormenor importante referido é o *overfitting* e apresenta duas formas de o combater: A primeira sugere que sejam feitas translação das imagens de *input*, tal como imagens em reflexo. A utilização de *Dropout* é a segunda estratégia que consiste em fixar como zero o *output* de cada neurónio que constitui uma camada escondida com probabilidade 0.5 (*Dropout* =0.5). Os neurónios são *dropped out* no sentido em que não contribuem para que a informação seja passada e como tal não participam no cálculo do gradiente nem na *backpropagation*. Portanto, por cada *input* é configurada na rede uma arquitetura diferente, porém todas as arquiteturas partilham os mesmos pesos e esta técnica leva que um neurónio em específico não possa depender de outros em particular e como consequência o algoritmo torna-se mais robusto visto que passa por mais arquiteturas até encontrar uma solução ótima.

Por fim, o que explorei neste trabalho final começou com as pesquisas de Sezer *et al* (2017, 2019).

Na pesquisa menos recente (2017) utiliza um algoritmo genético (*Evolutionary algorithm*) para optimizar os parâmetros de análise técnica (RSI e SMA) de forma a decidir a compra, venda ou a manutenção das ações do *Dow Jones 30*. Em seguida compara os resultados com a estratégia de *Buy and Hold* (BaH) e de uma forma geral o retorno anual produzido pelo algoritmo foi igual ou superior em metade das ações que constituem o índice.

[10]Gráfico ilustrativo da explicação no artigo de Krizhevsky, Sutskever & Hinton (2012)

[11]Indicadores técnicos utilizados: Indicadores de oscilação/*momentum*: RSI, William, CMO; Indicador de *trend*: SMA,EMA, WMA, HMA,TEMA, MACD. PPO; Compara preços correntes: CCI; Velocidade da mudança dos preços: ROC; Força e direção de *trend*: DMI e SAR – determina pontos de paragem ou mudança do preço.

Em *Algorithmic Financial Trading with Deep Convolutional Neural Networks: Time Series to Image Conversion Approach* (2019) vai mais de encontro ao apresentado neste trabalho. Neste caso em particular na conversão de imagens 1-D para imagens 2-D utilizam-se indicadores técnicos[11] como *input*.

Para a conversão da série temporal em imagens a duas dimensões foram escolhidos 15 indicadores técnicos, produzindo uma imagem 15x15, em que cada linha representa um indicador e cada imagem é rotulada como 1 (venda), 0 (mantem) ou 2 (compra). É comparado com BaH, RSI, SMA, LSTM e MLP. Obteve de forma geral melhores resultados do que qualquer outra das abordagens.

O modelo proposto, tanto usando dados do *Dow Jones* 30 ou de *Exchange Trade Funds* (*ETFs*) obteve uma melhor *performance* que a estratégia de *Buy and Hold*. Este apresentou uma taxa de sucesso de transações entre os 70% e 80% e o algoritmo conseguiu obter uma *accuracy* de 62% como o valor mais alto na fase de teste com os dados de *ETFs*. Os resultados também demonstraram que o modelo é consistente para diferentes períodos de tempo ou condições de mercado.

Para além dos trabalhos desenvolvidos de *Machine* e *Deep Learning* em diferentes áreas e em particular nos Mercados Financeiros referidos acima, a utilização de um modelo híbrido de redes neuronais é também uma solução presente na predição de Mercados Financeiros e tem apresentado bons resultados. Ding *et al* (2015) comprova a utilidade de *Deep Learning* na previsão do preço de ações e reflete a influência de eventos. Nesta investigação é utilizada uma rede neuronal em que os dados de entrada são palavras e expressões que provêm de notícias e o resultado são eventos. É através de *Neural Tensor Network* que capta a relação entre duas entidades, por exemplo, a primeira entidade seria os resultados trimestrais da EDP, a segunda entidade seria o lucro e o resultado produz um denominado *event embedding* – modelação utilizando *Natural Languangue Processing* (NPL) em que os eventos são mapeados para vetores de números reais.

Em seguida os eventos de longo e médio prazo são utlizados como dados de entrada na camada inicial numa *Convolutional Neural Network*. A utilização de CNN pretende combinar a influência de eventos de longo e médio prazo no movimento dos preços das ações. Em seguida a *feedforward neural network* constituída apenas por uma camada escondida seguida de uma camada de *output* que associa características globais captadas pela camada de *max pooling* à tendência de subida ou descida de uma ação. A precisão tanto na previsão de índice S&P500 como de ações em específico é melhor do que outros métodos (Rede Neuronal *standard*) [12].

Por último é feita uma simulação de mercado para testar a assertividade do modelo e o lucro que resulta do mesmo. A compra ou venda de ações acontece se a probabilidade de subida do preço for superior a 70% (*threshold* que foi definido) no caso de compra ou se a probabilidade de descida for superior aos mesmos 70% deve-se optar pela venda. A estratégia de *trading* utilizada também obteve maior lucro do que quando esta é utilizada com os outros modelos.

[12]Esquema da arquitetura da rede pode ser consultado no artigo de Ding (2015)

2. *Machine Learning* e Mercado Financeiros: Modelos de *Trading*

A Economia tal como a Matemática desenvolveram-se com a evolução do mundo e percebemos como este influencia a Economia e os Mercados Financeiros, tal como o inverso também é verdade. Para além disso percebemos que a Matemática ajuda a medir e prever essa influência e que isso torna o homem mais poderoso e capaz de aumentar tanto o seu conhecimento como também o seu lucro. É através dessa variação dos mercados e dos preços causadas pelos "espirros" do mundo ou por "twittes" ou notícias que se faz negócio baseado em crenças ou previsões no futuro, ou seja, que se vende ou se compra neste caso produtos financeiros (ações, obrigações, derivados ou taxas de câmbio) com a expectativa de ter lucro no futuro e isso é *Trading*.

A predição sempre foi um assunto relevante tal como o lucro que se obtém através dela e por isso é um fator atrativo tanto do ponto de vista do investidor como do investigador. Os modelos, estratégias ou análises antes da Inteligência Artificial ou da melhoria computacional eram baseados em regras estáticas e a análise acabava sempre por ficar desatualizada cedo demais. A acrescentar que neste tipo de área, existe também um risco operacional e o ser humano nem sempre toma a decisão mais racional se tiver outros motivos que influenciem a sua escolha, neste caso de compra ou venda.

A estratégia de *Buy and Hold* começou em meados dos anos 70 e tornou-se popular quando se procurava um investimento a médio ou longo prazo. De um modo simples, esta é uma estratégia mais conservadora em que essencialmente o investidor compra ações, obrigações ou até *ETFs* e mantem essa posição por longos períodos e portanto ignora as flutuações diárias dos mercados. Porém este tipo de estratégia requer "sangue frio" no que toca à volatilidade dos mercados e nem sempre a venda é feita no *timing* ideal.

A escolha em que produto investir é suportada muitas vezes por uma análise técnica ou fundamental.

Numa óptica mais conservadora e sem o auxílio de modelos de predição, a forma de analisar o mercado e qual será o *drive* ou o *flow* do mesmo é normalmente baseada nas duas abordagens referidas abaixo:

Análise Fundamental (1928) obtém informação detalhada acerca da empresa, como por exemplo, rácios de capital, *cash flows*, passivo e ativo e outros dados financeiros importantes da empresa.

Análise Técnica, focada no movimento do preço do ativo subjacente, baseando-se, portanto na Hipótese de Mercado Eficiente. Para o estudo desta variação são usados vários indicadores técnicos, *Relative Strength Index* (RSI), *Williams %R* e *Moving Average Convergence and Divergence* (MACD) são possivelmente os mais utilizados.

Relative Strength Index é um indicador de momento ou de oscilação que indica os momentos de maior queda e subida dos preços no passado. Ao mudar o preço, o RSI oscila entre 0 e 100. Se o valor é acima de 70 a ação está numa região de *overbought*, logo deve ser vendida. Enquanto se o valor estiver abaixo de 30 indica que está numa região de *oversold*, logo devemos comprá-la.

$$\boldsymbol{RSI} = 100 - \frac{100}{1 + \frac{média\ do\ ganho}{média\ de\ perda}}$$

Williams %R é também um indicador de momento que mostra condições ideias para a compra ou venda. E vai de encontro com a lógica do RSI. Oscila entre -100 e 0.

$$R = \frac{preço\ max - preço\ fecho}{preço \max - preço\ min}$$

Moving Average Convergence and Divergence indica a tendência ou a direção do preço da ação. Se a linha de MACD passa a linha de sinal numa direção ascendente, prevê-se que o preço suba se a linha estiver no sentido descendente, prevê-se que o preço desça.

Linha MACD: $Dia\ 12\ EMA - Dia\ 26\ EMA$

Linha de Sinal: $Dia\ 9\ EMA$

EMA - *Exponential Moving Average* - indica o movimento médio dos preços, dando maior peso aos dias mais recentes.

Porém, este tipo de análise apenas é um auxílio e não um método quando falamos de estratégias de compra ou venda de produtos financeiros.

Dentro do mundo da previsão temos Modelos Lineares, Modelos Não-Lineares, modelos de ML, entre outros.

Começando nos modelos lineares é sabido que o preço de ações, de índices bolsistas, *EFTs*, taxas de câmbio cotadas em bolsa são também séries temporais.

Uma série temporal (sucessão cronológica) é um conjunto de observações tomadas em instantes de tempo determinados com intervalos iguais.

Matematicamente, uma sucessão cronológica definida pelos valores $y_1, y_2, \ldots$ de uma variável y nos tempos $t_1, t_2, \ldots$ com $t_i = t_j$, ∀ i,j. Portanto, y é uma função de t simbolizada por $y = F(t)$

Por outro lado, um Processo Estocástico é qualquer família de variáveis aleatórias $\{X(t), t \in T\}$, em que T é o conjunto dos índices representativos do tempo.

Um processo estocástico $\{X(t), t \in T\}$ diz-se **fracamente estacionário ou estacionário** de 2ª ordem se ∀ t ∈ T, tem-se:

1. $E[X(t)] = \mu$

2. $Var[X(t)] = \sigma^2 < +\infty$

3. $Cov[X(t_1), X(t_2)] = \gamma(t_1, t_2) = \gamma(|t_2 - t_1|)\ \ t_1, t_2 \in T$

Portanto, uma série temporal diz-se *fracamente estacionária* se tiver média e variância constantes e finita ao longo do tempo, e a covariância entre dois valores da série depende apenas do valor da distância temporal entre os mesmos.

Uma série temporal tem quatro componentes importantes: Movimento de tendência, movimento cíclico, sazonalidade e movimento irregular ou aleatório. São estas componentes que determinam se uma série é estacionária ou não. Só partindo do pressuposto de ser estacionária é que podemos prever e interpretar relativamente a dados económicos.

Este tipo de série temporal tem algumas particularidades:

- **Independência** de eventos, ou seja, o que conheço hoje não me ajuda a conhecer o futuro (*Random Walk*).
- **Memória** longa, o passado reflete-se durante bastante tempo ou memória curta em que a dependência desaparece rapidamente.
- A sua **distribuição** apresenta caudas gordas, **ganhos e perdas assimétricas** tal como a presença de volatilidade.

Portanto, todas estas características tornam difícil que os modelos lineares "sirvam" a este tipo de séries temporais.

Devido a essa complexidade difícil de "normalizar", juntamente com os avanços feitos computacionalmente, têm-se vindo a elaborar modelos mais eficientes e assertivos e que entram então para o mundo da Inteligência Artificial, em particular do *Deep Learning*.

Por razões óbvias é pouco divulgado o que é utilizado efetivamente nos mercados do ponto de vista de *Machine Learning*. Existe um *gap* entre o trabalho académico (transparente e baseado em dados) e o trabalho de profissionais do meio (*blackbox* extremamente eficiente e focada no lucro).

Cavalcante *et al.* (2016) descreve trabalhos que têm sindo desenvolvidos na área de Mercados Financeiros desde 2009. De uma maneira resumida dos 56 artigos referidos, a maioria refere-se a predição de ações (36) ou índices bolsistas (25), só uma minoria (9) são utilizados para *trading*.

Num algoritmo, o *trading* é como uma segunda avaliação do mesmo. Enquanto a *accuracy* avalia a *performance* do algoritmo de um ponto de vista computacional ou matemático, o *trading* avalia se o algoritmo está pronto a ser usado no mundo real. Ou seja, é a avaliação do lucro gerado pelo modelo que permite que se confie no seu objetivo que é a predição. O retorno anual, índice de *Sharpe*, máximo e mínimo de lucro ou perda, entre outras são algumas das métricas utilizadas nessa avaliação.

Muitos dos modelos ou algoritmos referidos, no seu processo de desenvolvimento foram beber a conceitos básicos da análise financeira. É usado tanto a análise técnica como a fundamental para a predição. Enquanto as variáveis técnicas costumam ser usadas para *short term trading*, as variáveis fundamentais costumam ser usadas para *long term trading*.

A seleção, extração ou filtro das variáveis mais relevantes é uma parte fundamental da preparação do algoritmo. Nos trabalhos que têm sido desenvolvidos verificou-se que mais do que o passado são as notícias que influenciam os preços e que podem então ser consideradas uma ajuda na predição.

Esta pesquisa refere que no âmbito de ANN, o algoritmo mais utilizado é o de MLP. O uso desta rede permite que seja aprendida a relação entre indicadores técnicos de forma a prever máximos e mínimos diários do preço das ações, como por exemplo, o preço de fecho das ações do *Indian Stock Exchange*. Outro exemplo, (González *et al.* 2011) de uma *Feedfoward Neural Network* que aprende o indicador RSI de algumas ações cotadas no IBEX 35. E o RSI predito é utilizado no sistema de *trading* que indica ou aconselha quando devem ser compradas ou vendidas as ações analisadas.

São também referidos modelos híbridos e da sua eficácia numa óptica de "dividir para conquistar".

Por fim e de uma forma geral os estudos feitos em *Machine* e *Deep Learning* na predição nos Mercados Financeiros e em particular em estratégias de compra e venda utilizam na sua maioria: Índices bolsistas, visto que são valores que traduzem a tendência geral de um determinado conjunto de títulos, nomeadamente ações, através de um média das subidas e das descidas, como também *Exchange Traded Fund*, pois é um conjunto diversificado de ativos que transaciona em Bolsa (como uma ação) e são das formas mais eficiente e populares de investimento.

3. Análise Empírica

Neste capítulo será descrita a parte experimental deste trabalho, que passará pelos seguintes pontos:

- Método utilizado para a construção dos dados de *input* e arquitetura da rede;
- Resultados e Avaliação do ponto de vista computacional como também de eficiência no processo de *Trading*.

Os dados utilizados correspondem ao índice *Dow Jones 30* – Microsoft, e para a análise foi considerado o preço de fecho desde 1987 até 2020[12] (janela temporal de 33 anos, dados diários, 5dias/semana), o que corresponde a um total de 8518 dias[13]. Estes dados foram retirados da plataforma *Yahoo Finance*.

Esta análise foi feita através da linguagem de programação *Python*, em particular usando o IDE *Spyder* e as bibliotecas *Tensorflow* e *Keras* para poder usar a rede neuronal em específico.

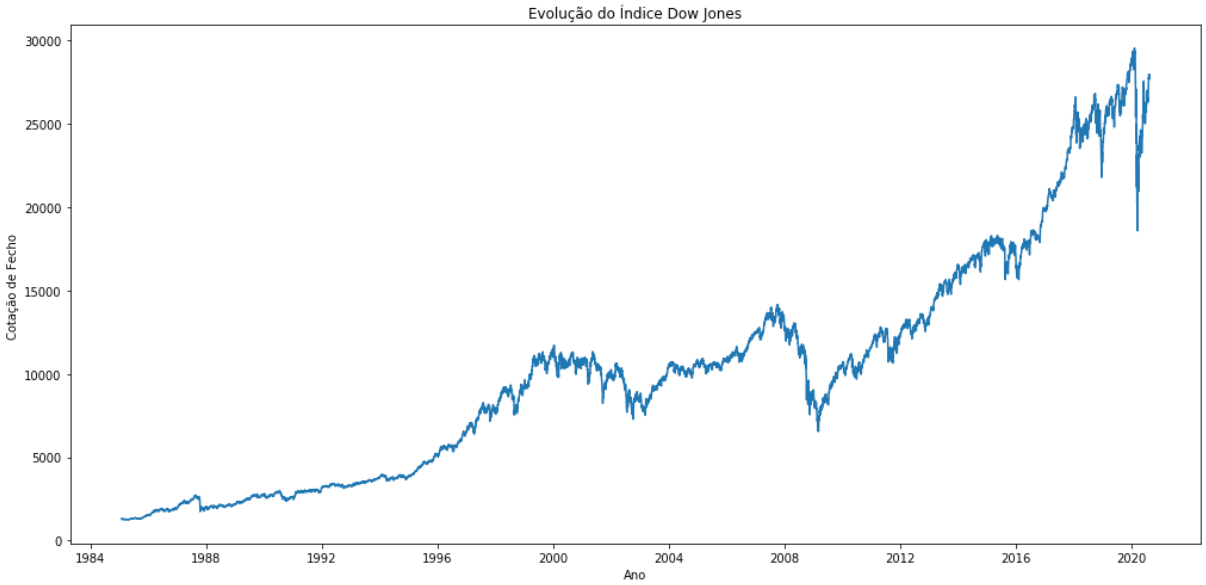

Figura 6: Índice *Dow Jones*

[12]Dados retirados até dia 15 de Outubro de 2020.

[13]Dados originais de 1997 até 2017 (janela temporal de 20 anos, dados diários, 5dias/semana), o que corresponde a um total de 5035 dias.

3.1. Método

Tal como referido inicialmente este trabalho debruçou-se sobre uma abordagem não convencional para a previsão de ações, em particular, previsão de pontos de *trading* das mesmas, que usa uma rede neural convolucional para determinar cenários de *Buy*, *Sell* ou *Hold* através de imagens. Neste caso a série temporal é convertida numa série de imagens, cada uma destas corresponde a um gráfico de barras dos preços das ações. O objetivo é encontrar os pontos mais adequados na série temporal dos preços da ação associada para transações de *buy-sell* de forma a maximizar o lucro do algoritmo de *trading*.

O método proposto tem as seguintes fases principais:

- Extração / Transformação do conjunto de dados;
- Construção da imagem (a partir da série temporal);
- Rotulagem (*Labeling)* de cada imagem;
- Uso da CNN para previsão;
- Avaliação e interpretação dos resultados.

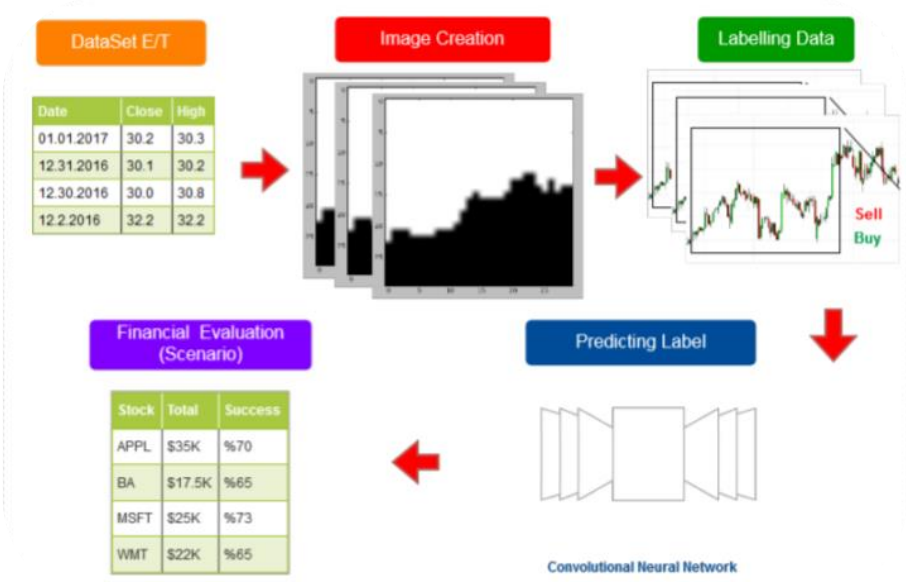

Figura 7: Método Proposto[14]

[14] Imagem original do artigo de Sezer & Ozbayoglu (2019)

Os dados, como foi referido acima, representam os preços da Microsoft desde 1987 até 2020. Estes dados foram divididos em dois conjuntos mais pequenos, o primeiro de 1987 a 2010 (6053 dias) e o segundo de 2011 até 2020 (2465 dias). O primeiro conjunto corresponde aos dados de treino e representa 80% do nosso universo, enquanto os restantes 20% que resultam no segundo conjunto correspondem aos dados de teste. Todos estes preços estão em ficheiros com formato *csv*.

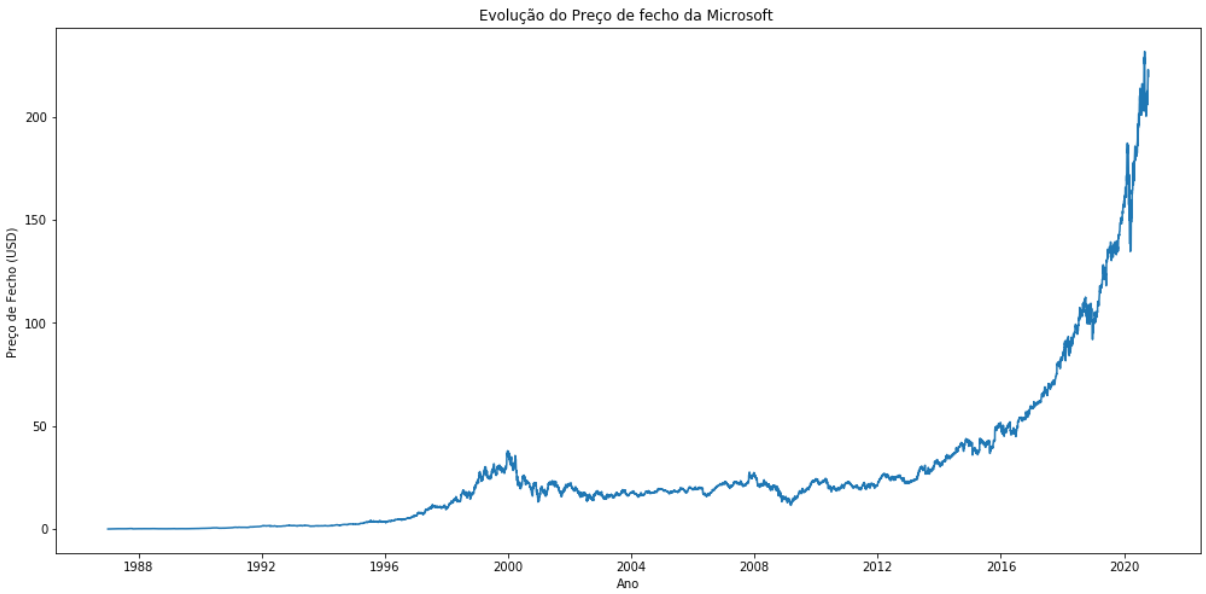

Figura 8: Preço de Fecho da Microsoft

De seguida são contruídos os gráficos de barras com os preços e que serão as imagens de entrada para a rede (o *input*). Cada imagem convertida contém 30 dias de preços da *stock* (eixo do y) e tempo (eixo do x), que resulta numa imagem de dimensão 30x30 pixéis.

Na criação das imagens para em seguida as rotular é necessário tratar os dados, ou seja, será necessário normalizar os dados considerados.

O processo da formação das imagens e o *labeling* das mesmas acontecem quase em simultâneo. Cada imagem pode ter como rótulo 0, 1 ou 2, que corresponde a *Hold*, *Buy* ou *Sell* respectivamente. Esta classificação é feita através do declive entre os pontos/preços nos dias considerados. (Anexo A)

Primeiro calcula-se o declive que servirá de referência e comparação para o declive corrente das imagens. Este consiste na diferença entre o valor do dia 33 e dia 30, dia 31 e 34 e assim sucessivamente. Através deste total de declives de todas as futuras imagens consegue-se estimar o declive médio de forma a encontrar os valores (declives) de separação que irão permitir fazer a classificação de cada imagem tendo em conta o declive instantâneo da mesma.

$$Declive\ de\ Referência[1, \dots, n] = \frac{preço\ dia\ 33 - preço\ dia\ 30}{dia\ 33 - dia\ 30}$$

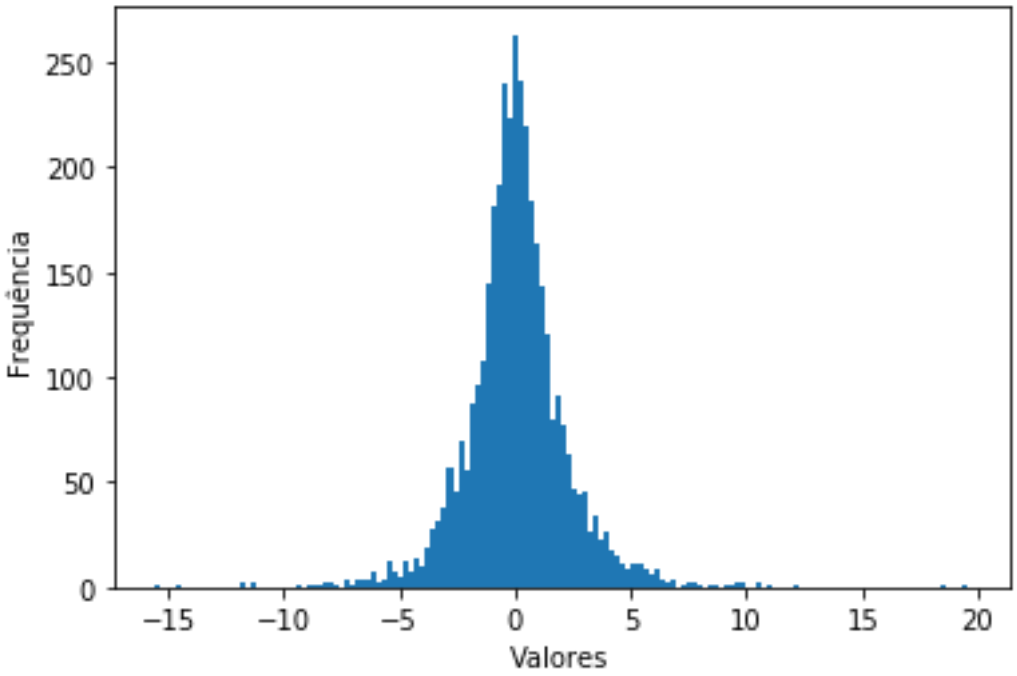

Figura 9: Histograma com distribuição Gaussiana dos valores da lista dos declives de referência

Apenas na primeira imagem o declive corrente[15] é resultado do último dia que a compõe e do dia 33 (3º dia da imagem a seguir). Para as restantes imagens, estas são rotulados tendo em conta o primeiro dia que a constitui e o 3º dia a seguir. Desta forma iterativa haverá um número menor de imagens, mas de resto todos os dias e respetivos preços são classificados.

$$Declive\ Corrente[1, \dots, n] = \frac{preço\ dia\ 33 - preço\ dia\ 30}{dia\ 33 - dia\ 30}$$

[15]No artigo de Sezer & Ozbayoglu (2019) o declive corrente para a classificação de uma imagem foi calculado com um intervalo de 15 dias, ou seja, era calculado entre o dia 45 e dia 30, dia 46 e dia 31 e por aí adiante.

Rotulagem da Tendência é definida para cada imagem como mostra a equação abaixo[16]:

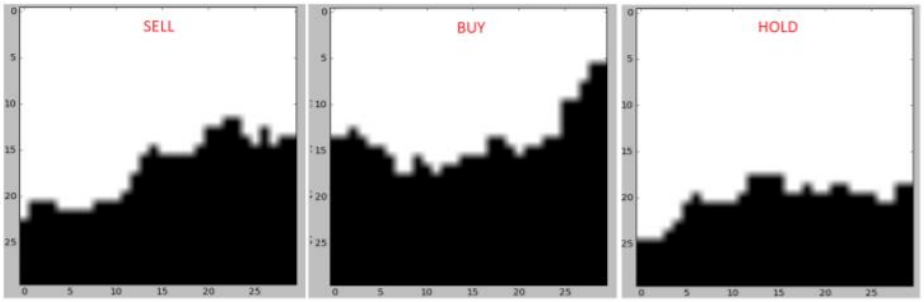

Figura 10: Exemplos de imagens 30x30 e respectivo rótulo[17]

$$l = \begin{cases} 1 = (\text{Comprar}), se\ Declive\ Corrente > Declive\ de\ Referênica[{}^{3n}/_{5}] \\ 0 = (\text{Manter}), se\ Declive\ de\ Referênica\left[{}^{2n}/_{5}\right] < Declive\ Corrente \\ \qquad\qquad e\ Declive\ de\ Referênica\left[{}^{3n}/_{5}\right] > Declive\ Corrente \\ 2 = (Vender), se\ Declive\ Corrente < \ Declive\ de\ Referênica\left[{}^{2n}/_{5}\right] \end{cases}$$

Ou seja, se na imagem em análise se verificar uma tendência decrescente está-se perante uma situação de compra (imagem com rótulo 1), caso contrário se se observar uma tendência crescente é uma situação de venda (imagem com rótulo 2). Se nenhum destes casos se verificar, ou seja, não se verifica uma tendência, nada acontece (imagem com rótulo 0).

[16]Para os valores de referência para a classificação das imagens consoante os valores declive (3n/5 e 2n/5), o numerador refere-se ao número de classes (2n- 2 classes e 3n-3 classes), ou seja, 1 e 2 ou 0, 1 e 2 e denominador representa os 5 dias de trabalho por semana.

[17] Imagem original do artigo de Sezer & Ozbayoglu (2019)

De forma a clarificar esta metodologia segue um exemplo simples:

255	255	255																									
255	255	255																									
255	255	255																									
255	255	255																									
255	255	255																									
255	255	255																									
255	255	255																									
255	255	255																									
255	255	255																									
255	255	255																									
255	255	255																									
255	255	255																									
255	255	255																									
255	255	255																									
255	255	255																									
255	255	255																									
255	255	255																									
255	255	255																									
255	255	255																									
255	255	0																									
255	0	0																									
255	0	0																									
255	0	0																									
255	0	0																									
0	0	0																									
0	0	0																									
0	0	0																									
0	0	0																									
0	0	0																									
0	0	0																									

Figura 11: Exemplo da construção de uma imagem 30x30

Para os preços do dia 31 ao 61 é então criada uma imagem com dimensão 30x30 (ilustrado na Figura 11). Se o preço para o dia 31 tivesse o valor de USD7, esse valor seria convertido para uma escala de 30x30 (dimensão da imagem) e iria ser criada uma barra na imagem. E assim sucessivamente até estar completa a imagem. Essa imagem terá como declive do dia 31 ao dia 34 e será comparado com o declive de referência para os pontos de "Buy", "Hold" ou "Sell".

Os valores a zero correspondem às barras enquanto os pixéis "a vazio" são preenchidos com o valor 255. Ou seja a primeira barra preenchida com os valores a "0" corresponde ao preço USD7. O espaço ainda a vazio será preenchido usando a mesma lógica e ficando assim criada a imagem que representa os preços desde o dia 31 ao 61.

Depois, a etiqueta e a sua imagem associada são combinados num ficheiro *csv* para a fase de treino e de teste. Cada linha do ficheiro corresponde a uma imagem, sendo o penúltimo valor da linha o preço do dia e o último valor a etiqueta da imagem. No exemplo acima, esta linha do ficheiro teria uma matriz 30x30 em que cada linha da matriz era disposta uma seguida a outra horizontalmente, cobrindo os preços do dia 31 ao 61 e teria o preço corrente do dia 31 (USD7) e *label* = 2.

Tal como referido anteriormente este trabalho foi também ao encontro de melhorar o modelo apresentado, deste modo alterou-se o cálculo do declive, pois os autores sugeriram que este poderia ser um ponto a melhorar. Visto que estamos a falar de ações bastante voláteis não faz sentido considerar declives com pontos que distam 15 dias. Por isso considerou-se declives entre um número menor de dias futuros. Tanto o declive de referência como o declive corrente são calculados para a mesma janela temporal de dias, pois só assim seria possível comparar as duas realidades.

Número de Imagens com o Rótulo	
=0 ("Hold")	1197
=1 ("Buy")	2405
=2 ("Sell")	2406
Número Total de Imagens de Treino: 6008	

Tabela 1: Dados de treino

Rácio das Imagens Rotuladas- Conjunto de Treino	
=0 ("Hold")/1("Buy")	0.5
=0 ("Hold")/2 ("Sell")	0.5
=1 ("Buy")/2 ("Sell")	1

Tabela 2: Rácio de Imagens de Treino

Número de Imagens com o Rótulo	
=0 ("Hold")	688
=1 ("Buy")	942
=2 ("Sell")	789
Número Total de Imagens de Teste: 2419	

Tabela 3: Dados de teste

Rácio das Imagens Rotuladas- Conjunto de Teste	
=0 ("Hold")/1("Buy")	0.7
=0 ("Hold")/2 ("Sell")	0.9
=1 ("Buy")/2 ("Sell")	1.2

Tabela 4:Rácio de Imagens de Teste

É de notar que as imagens rotuladas com zero estão em menor número e mesmo utilizando outros dados e outros horizontes temporais esta é uma característica difícil de contrariar. Porém, se se duplica-se o número de imagens com esta classificação ou se se tenta-se de outras formas aumentar este valor o resultado seria adulterar a realidade dos mercados. É um facto comum e bastante conhecido que nos Mercados Financeiros e se considerarmos *trading* de curto, médio ou mesmo longo prazo a posição de "*Hold*" é sempre a que existe com menor frequência. Em alturas de *Bear Market*[18] é possível que este acontecimento se verifique com maior frequência, porém ao se utilizar uma janela temporal longa o objetivo é captar todas essas tendências e movimentos do mercado.

[18] Termo utilizado para caracterizar um momento negativo dos mercados, em que os preços caem em mais de 20%. E são por isso geralmente acompanhados por um sentimento negativo do investidor, tal como a perspectiva de declínio da Economia.

Na fase de aprendizagem é construída e ajustada a arquitetura da CNN para treinar da melhor forma possível os dados mencionados.

Neste caso em concreto optou-se pela seguinte **Arquitetura da Rede Neuronal Convolucional[19]:**

- Camada de *Input*;
- 2 camadas convolucionais (1 de 30x30x32 e 1 de 30x30x64);
- 1 camada de *Max pooling*;
- 1 camada de *Dropout* (0.25);
- Camada totalmente conexa;
- 1 camada de *Dropout* (0.50);
- Camada de *Output.*

O tamanho do filtro é optimizado escolhendo um filtro de tamanho 3x3. Na literatura diferentes tamanhos de filtros são adotados nas CNNs: 3x3, 5x5 e 7x7. Diminuir o tamanho do filtro geralmente resulta em apanhar mais detalhes das imagens. Na proposta, preferiu-se filtro 3x3 devido ao relativo pequeno tamanho das imagens usadas como *input*.

A rede CNN é implementada usando as bibliotecas *Keras*, *Tensorflow* e cada *test run* dura por 50 *epochs* e tem lote/*batch size* de 1024 imagens.

Em relação à arquitetura inicial e para fazer face a problemas de *overfitting* e melhorar a precisão do modelo tal como os novos declives e novos dados apenas se diminui o número de *epochs* de 100 para 50.

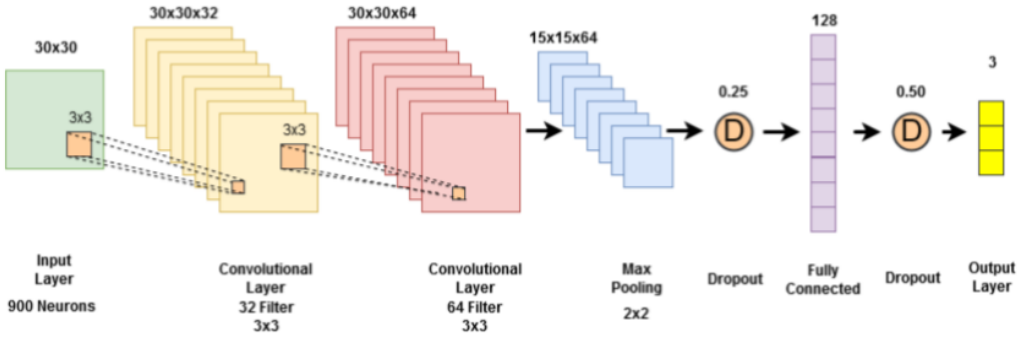

Figura 12: Arquitetura típica para CNN[20]

[19] Algoritmo sumarizado para o modelo proposto pode ser consultado no artigo de Sezer & Ozbayoglu (2019)

[20] Imagem original do artigo de Sezer & Ozbayoglu (2019)

3.2 Avaliação dos Resultados

De forma a avaliar se o modelo e a rede propostas se adequam aos dados e ao objetivo deste trabalho final é necessário separar a avaliação da rede da avaliação do modelo como um todo. Pois enquanto a avaliação da rede tem em vista a sua precisão (valor real *versus* valor predito), a avaliação do modelo tem em consideração o lucro que este algoritmo gera. E o que se pretende por último não é o algoritmo mais preciso, mas o que atinja um maior lucro através da compra e venda de ativos financeiros.

A) Avaliação da Rede

Para avaliar a *performance* da rede utilizada face aos dados de treino e de teste foram consideradas as seguintes métricas:

***Accuracy*/Precisão:** Corresponde ao rácio entre o número correto de valores preditos face ao número total de exemplos.

$$Precisão = \frac{Número\ de\ valores\ preditos\ corretos}{Número\ total\ de\ valores\ preditos}$$

Log Perda/Perda Logarítmica: Esta função penaliza as classificações erradas e é utilizada para avaliar a classificação por classes. Ao trabalhar com Log Perda, o classificador deve atribuir probabilidade a cada classe para todas as amostras. Supondo que existem N amostras pertencentes a M classes, então a Log Perda é calculada pela seguinte fórmula:

$$Perda\ Logarítmica = \frac{-1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij} * \log(p_{ij})$$

y_{ij}, indica se a amostra i pertence à classe j ou não

p_{ij}, indica a probabilidade da amostra i pertencer à classe j

Quanto mais próximo estiver este valor de zero, maior é a precisão.

Erro absoluto médio (MAE):

$$\mathbf{MAE} = \frac{1}{N}\sum_{i=1}^{N} |y_i - \widehat{y}|$$

Erro quadrático médio (MSE)

$$\mathbf{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2$$

Onde:

y_i, valor observado

$\hat{y}$, valor predito para y

***Confusion Matrix*/Matriz Confusão:** As colunas desta matriz correspondem ao valor actual e as linhas ao valor predito. Terá tantas linhas e colunas quantas classes de classificação existerem.

Neste caso há três classes (0,1 e 2), logo forma-se uma matriz 3x3. Por exemplo, a 1ª linha e a 1ª coluna corresponde ao número de vezes em que o valor actual foi igual ao valor predito para classe com *label* = 0. O ideal é ter apenas valores na diagonal principal e poucos valores nas outras linhas e colunas, pois isso significa que a classe foi predita corretamente na maior parte dos exemplos de treino e de teste.

		Valor actual		
Valor predito	Etiqueta	**0**	**1**	**2**
	0	70	10	10
	1	5	80	5
	2	1	4	85

Tabela 5: Exemplo ilustrativo de uma confusion matrix para classificação das 3 classes

Através desta classificação é possível avaliar "Recall", "Precision" e "F1-score" de cada classe. Para tal os dados são divididos em quatro categorias: Verdadeiro Positivo (VP, o modelo previu positivo e é verdade), Verdadeiro Negativo (VN, o modelo previu negativo e é verdade), Falso Positivo (FP, o modelo previu positivo e é falso), Falso Negativo (FN, o modelo previu negativo e é falso). (O Verdadeiro e Falso são sempre referentes ao valor actual e Positivo e Negativo são sempre referentes ao predito).

		Valor actual	
Valor predito	Classe	**Positivo**	**Negativo**
	Verdadeiro	VP	FN
	Falso	FP	VN

Tabela 6:*Confusion Matrix* por classe

$$\boldsymbol{Recall} = \frac{VP}{VP + FN}$$

$\boldsymbol{Recall}$- Representa a percentagem de vezes em que os valores actuais corresponderam ao predito;

$$\boldsymbol{Precision} = \frac{VP}{VP + FP}$$

$\boldsymbol{Precision}$- Representa quanto dos valores preditos corresponderam aos actuais.

$$\boldsymbol{F1Score} = \frac{2 \times Recall \times Precision}{Recal + Precision}$$

F1-Score- corresponde à média harmónica de *recall* e *precision*.

Resultados da Fase de Treino:

Rótulo	**0**	**1**	**2**
0	**39**	740	418
1	85	**1344**	976
2	87	1429	**890**

Tabela 7: *Confusion Matrix* - Fase de Treino

Rótulo	**Precisão**	**Recall**	**F1-Score**	**Total**
0	18%	3%	6%	1197
1	38%	56%	45%	2405
2	39%	37%	38%	2406
Accuracy			**38%**	**6008**

Tabela 8: Avaliação da fase de treino

Resultados da Fase de Teste:

Rótulo	**0**	**1**	**2**
0	**25**	579	84
1	16	**770**	156
2	23	651	**115**

Tabela 9: *Confusion Matrix* - Fase de Teste

Rótulo	**Precisão**	**Recall**	**F1-Score**	**Total**
0	39%	4%	7%	688
1	39%	82%	52%	942
2	32%	15%	20%	789
Accuracy			**38%**	**2419**

Tabela 10: Avaliação da fase de teste

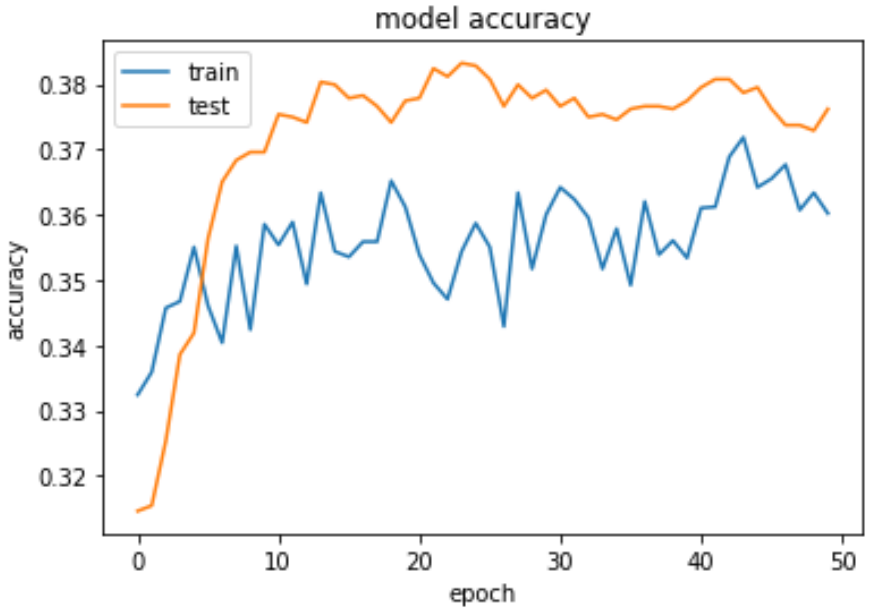

Figura 13: *Accuracy* do modelo- Fase de treino *vs* Fase de teste

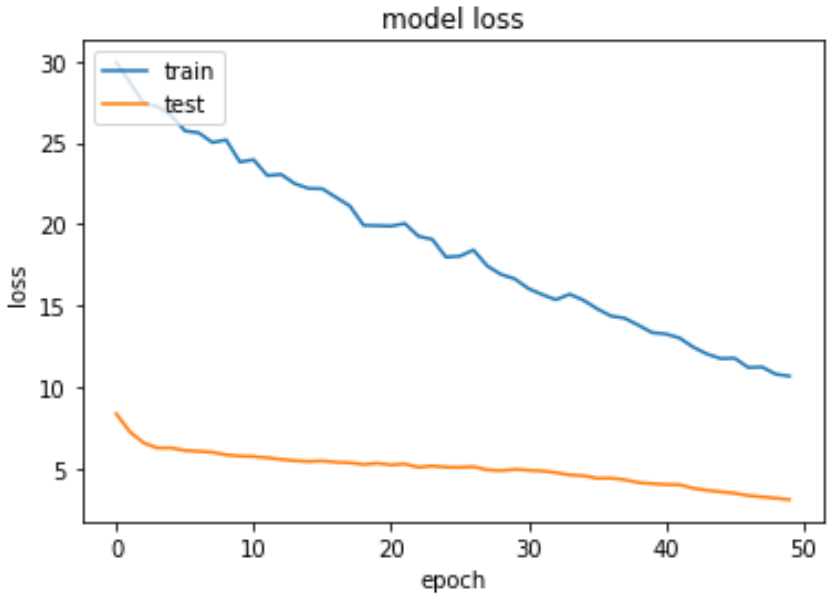

Figura 14: *Loss* do modelo- Fase de treino *vs* Fase de teste

Apesar da *accuracy* (38% na fase de treino e de teste *vs* 90% na fase de treino e 46% na fase de teste) não verificar um valor muito alto (inferior ao apresentando por Sezer & Ozbayoglu (2019)- Tabela 16 e 18), neste primeiro contributo o objetivo era que o algoritmo fosse capaz de classificar de forma correta ou pelo menos mais consistente/homogénea as 3 classes, por isso o foco centrou-se mais nos resultados da *Confusion matrix* do que no valor da *accuracy*. Pois pelos testes feitos, um valor de *accuracy* mais alto refletia que umas das classes era classificada corretamente em um número elevado de vezes, porém

o mesmo não se verificava com as outra classes, nomeadamente quando o rótulo da imagem era 0 (visto que este existe em menor proporção), ou seja, para este modelo em específico um valor mais alto de *accuracy* "escondia" também os seus pontos fracos.

De salientar que na fase de teste face à fase de treino, o algoritmo conseguiu melhorar na classificação de imagens com os rótulos de 0 e 1, como também foi possível reduzir o *overfitting* que existia originalmente (*accuracy* da fase de treino muito superior à de teste). Esta é uma fase decisiva do modelo, ao replicar o modelo inicial obteve-se resultados muito semelhantes aos de Sezer & Ozbayoglu (2019) porém, ao alterar os declives, a fase de treino obteve melhores resultados, em contrapartida foi difícil retirar o frequente *overfitting* na fase de teste. De uma forma geral, após algum esforço e uma fase de tentativa de erro para tentar ajustar o algoritmo, para uma primeira abordagem, não obstante de existir muito que possa ser explorado e melhorado, os resultados foram satisfatórios.

Resultados obtidos com o modelo original

Dados originais:

Número de Imagens com o Rótulo	
=0 ("Hold")	346
=1 ("Buy")	1855
=2 ("Sell")	1590
Número Total de Imagens de Treino: 3791	

Tabela 11: Modelo original - Dados de treino

Rácio das Imagens Rotuladas- Conjunto de Treino	
=0 ("Hold")/1("Buy")	0.2
=0 ("Hold")/2 ("Sell")	0.2
=1 ("Buy")/2 ("Sell")	1.2

Tabela 12: Modelo original - Rácio de Imagens de Treino

Número de Imagens com o Rótulo	
=0 ("Hold")	154
=1 ("Buy")	652
=2 ("Sell")	407
Número Total de Imagens de Teste: 1213	

Tabela 13: Modelo original - Dados de teste

Rácio das Imagens Rotuladas- Conjunto de Treino	
=0 ("Hold")/1("Buy")	0.2
=0 ("Hold")/2 ("Sell")	0.4
=1 ("Buy")/2 ("Sell")	1.6

Tabela 14:Modelo original - Rácio de Imagens de Teste

Resultados da Fase de Treino:

Rótulo	**0**	**1**	**2**
0	**241**	87	18
1	1	**1843**	11
2	4	261	**1325**

Tabela 15: Modelo original- *Confusion Matrix* - Fase de Treino

Rótulo	**Precisão**	**Recall**	**F1-Score**	**Total**
0	98%	70%	81%	346
1	84%	99%	91%	1855
2	98%	83%	90%	1590
Accuracy			**90%**	**3791**

Tabela 16: Modelo original -Avaliação da fase de treino

Resultados da Fase de Teste:

Rótulo	**0**	**1**	**2**
0	**6**	105	43
1	21	**467**	164
2	10	315	**82**

Tabela 17: Modelo original - *Confusion Matrix* - Fase de Teste

Rótulo	**Precisão**	**Recall**	**F1-Score**	**Total**
0	16%	4%	6%	154
1	53%	72%	61%	652
2	28%	20%	24%	407
Accuracy			**46%**	**1213**

Tabela 18: Modelo original - Avaliação da fase de teste

B) Avaliação do modelo de *Trading*

Na última fase, a *performance* do modelo de *trading* pela estratégia de transação *Buy-Sell*, os resultados são avaliados usando métricas de avaliação financeira. Cada ação é comprada, vendida ou mantida de acordo com o rótulo predito.

Cenário de avaliação financeira (S):

$$S = \begin{cases} Número\ de\ ações = \dfrac{Capital\ Total}{Preço}, & \text{se rótulo} = "Comprar" \\ Sem\ ação, se\ \text{rótulo} = "Manter" & \\ Total\ de\ Capital = Número\ de\ ações\ \times Preço, se\ \text{rótulo} = "Vender" & \end{cases}$$

Para testar a estratégia considerou-se um capital inicial para a avaliação financeira de USD10.000 e comissão de USD1 por transação.

A métrica de avaliação seguinte foi a adoptada para este estudo:

Annualized Return/ Retorno Anual através da CNN (**AR**)

$$\boldsymbol{AR} = \frac{Capital\ Total^{\frac{1}{nº\ anos}}}{Capital\ Inicial} \times 100$$

Annualized number of transactions/Número anual de transações (**AnT**)

$$\mathbf{AnT} = \frac{Total\ de\ Transações}{Nº\ anos}$$

Percentage of success/Percentagem de sucesso (**PoS**)

$$\mathbf{PoS} = \frac{Total\ de\ transações\ bem\ sucedidas}{Total\ de\ transações} \times 100$$

Average percent profit per transaction/Percentagem média de lucro por transação (**ApT**)

$$\mathbf{ApT} = \frac{Percentagem\ total\ de\ lucro}{Total\ de\ transações} \times 100$$

Average transaction length/ Comprimento médio de transações (**L**)

$$\mathbf{L} = \frac{Total\ do\ comprimento\ das\ transações}{Total\ de\ transações} \times 100$$

Maximum profit percentage in transaction/ Percentagem máxima de lucro numa transação (**MpT**)

Maximum loss percentage in transaction /Percentagem máxima de perda numa transação (**MlT**)

Maximum capital / Capital máximo (**MaxC**)

Minimum capital/ Capital mínimo (**MinC**)

Idle Ratio (**IdleR**)

$$\mathbf{IdleR} = data.lenght^{21} - \frac{Total\ do\ comprimento\ das\ transações}{data.lenght^{21}} \times 100$$

Quanto menor o seu valor maior é o número de transações. O total do comprimento das transações é maior visto que representa a magnitude do número de transações feitas num intervalo espacial. Quanto mais espaçado for o intervalo entre o ponto de venda e compra maior é esse intervalo, mas menor serão o número de transações.

Average Percentage of Maximum Drawdown/ Percentagem média da maior queda (**MDD**)

Return Over Maximum Drawdown / Retorno após a maior queda (**RoMaD**)

[21] Corresponde o número total de pontos de trading preditos pela rede neuronal

Esta parte apenas se foca nos resultados obtidos por Sezer *et. al* (2019).

Dois períodos de treino e de teste foram escolhidos para representar condições de mercado diferentes.

No primeiro caso, o modelo foi treinado usando os preços dos constituintes do *Dow Jones 30* entre 1997-2007 e o período de teste foi entre 2007- 2012 que inclui a crise financeira de 2008. Durante este período particular de tempo, o mercado de ações fez violentos movimentos/*swings* resultando no período de volatilidade elevada.

No segundo caso, o modelo foi treinado com dados entre 1997-2012 e os dados de teste foram entre 2012-2017. Este foi um período de *Bull Market*[22] com volatilidade relativamente baixa.

Numa primeira análise, uma amostra de transações da JPM durante 2007 a 2012 resultou num ganho de USD15.871,19 (face ao capital inicial de USD10.000) consequência de as transações de ganho terem sido superiores em número e montante em relação às situações de perda (Tabela 19). E para este *stock* em particular, o modelo proposto superou os resultados da estratégia de *Buy and Hold* (Figura 15).

Número da Transação	Intervalo (Dias)	Ganho (USD)	Capital (USD)
1	21-25	-561,19	9 481,81
16	117-117	60,62	9 120,70
40	300-301	415,05	10 747,49
60	402-403	1 666,47	14 833,34
80	537-543	2 545,37	17 889,30
100	680-683	1 385,12	23 095,23
120	806-807	558,66	23 167,22
140	892-893	-442,64	21 281,64
160	1029-1030	-364,75	19 875,25
180	1168-117	1 099,19	19 840,19
187	1211-1213	2 351,19	25 871,19

Tabela 19: Amostra de Transações da JPM (2017-2012)[23]

[22] Corresponde a um período de tempo nos mercados financeiros em que o preço de um ativo ou título sobe continuamente. Os *traders* empregam uma variedade de estratégias, como por exemplo BaH, para lucrar com os mercados em alta.

[23] Imagem e tabela original do artigo de Sezer & Ozbayoglu (2019)

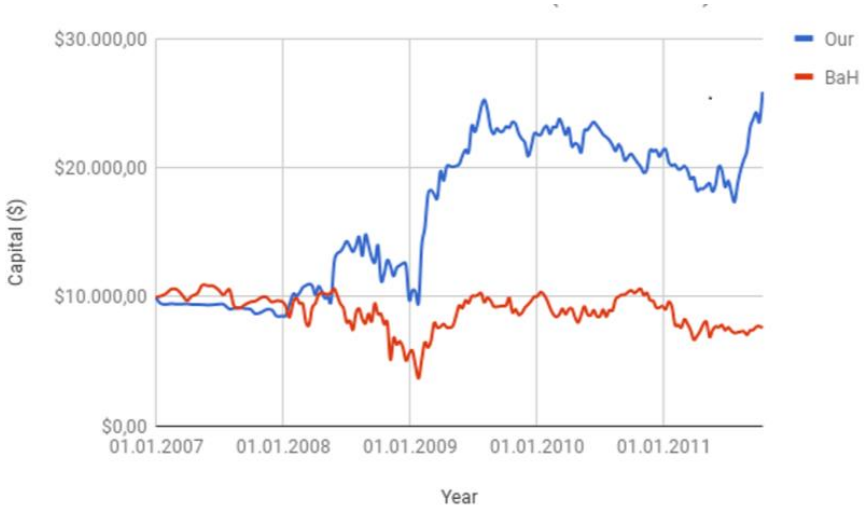

Figura 15: Lucro JPM (2007-2012)[23]

Para todos as *stocks*[24] do *Dow Jones* foi calculado o retorno anual (AR) para ambos os casos e para ambas as estratégias de *trading* (algoritmo proposto e BaH).[25]

Modelo	**Média**	**Desvio- Padrão**
CNN-BI(2007-2012)	7,20%	10,56%
BaH(2007-2012)	5,86%	9,67%
CNN-BI(2012-2017)	5,84%	6,30%
BaH(2012-2017)	13,25%	6,97%

Tabela 20: Retorno das Estratégias em estudo[25]

	Stock	
Modelo	Maior AR	Menor AR
CNN-BI(2007-2012)	APPL (37,46%)	GS (-18,86%)
BaH(2007-2012)	APPL (35,47%)	GS (-12,71%)
CNN-BI(2012-2017)	MMM (14,40%)	IBM (-8,6%)
BaH(2012-2017)	HD (26,13%)	IBM (-0,57%)

Tabela 21: Melhor *vs* Pior *Stock Perfomance*[25]

[24]Foram consideradas 29 ações e não as 30 que constituem o Índice *Dow Jones*, porque a Visa não tinha pontos suficientes para os períodos em estudo considerados.

[25]Avaliação em detalhe no artigo de Sezer & Ozbayoglu (2019)

O modelo de forma geral foi melhor do que BaH no primeiro caso (2007-2012) com um retorno anual foi 7,2% e 5,86% respectivamente. Com o algoritmo da CNN, considerando o universo das 29 *stocks* apenas 9 apresentaram perdas face a uma perda de 10 da estratégia de BaH. A Apple obteve a melhor performance em ambas as estratégias (37,46% *vs* 35,77%).

Durante o segundo caso de treino (1997-2012) e teste (2012-2017), o modelo de CNN não foi capaz de ser melhor do que BaH, devido ao facto do mercado estar num estado de *bull market* (preços à espera de subir) e não sofreram nenhum contratempo significativo por um longo e interrupto período de tempo. Perante estas condições de mercado geralmente não é fácil ser melhor que estratégia BaH. No período de 2012- 2017, o Retorno anual da CNN foi de 5,84% enquanto BaH apresentou um AR de 13,25%. Neste cenário o algoritmo perdeu 7 vezes em 29, enquanto BaH perdeu apenas 2. A 3M Company apresentou o máximo retorno de 14,40% (CNN), em comparação com um retorno máximo de 26,13% por parte da IBM (BaH).

Métricas de Avaliação	**Média (2007-2012)**	**Média (2012-2017)**
Retorno Anual(**AR**)	7,20%	5,84%
Número anual de transações (**AnT**)	52,70	53,44
Percentagem de sucesso (**PoS**)	52,35%	53,41%
Percentagem media de lucro por transação (**ApT**)	0,23%	0,18%
Comprimento médio de transações (**L**)	3,11	3,00
Percentagem máxima de lucro numa transação (**MpT**)	17,30%	8,31%
Percentagem máxima de perda numa transação (**MlT**)	-12,59%	-8,07%
Capital máximo (**MaxC**)	USD 15 767,53	USD 14 446,25
Capital minimo (**MinC**)	USD 7 395.77	USD 8 778,68
Idle Ratio (**IdleR**)	48,20%	48,88%
Percentagem média da maior queda (**MDD**)	34,92%	19,48%
Retorno após a maior queda (**RoMaD**)	121,31%	61,62%

Tabela 22: Estatísticas de *Perfomance* do Modelo CNN-BI[26]

Através das métricas referidas acima para a avaliação do modelo do ponto de vista de *Trading* para os dois casos em estudo os resultados da Tabela 22 mostram que a forma como o modelo reage a diferentes condições de mercado não muda muito (AR, AnT, PoS,L, MaxC, MinC e IdleR). Os resultados são um pouco mais satisfatórios no período de 2007 a 2012.

O número de transações de sucesso foi de 19 em 29 e 21 em 29, nos períodos 2007-2012 e 2012-2017 respetivamente.

Em ambos os períodos de teste foi feita em média uma transação de 3 em 3 dias e um número médio de transações por ano que indica que pelo menos o modelo foi capaz de gerar transações de *buy-sell* quase uma vez por semana mostrando assim consistência do algoritmo para ambos os períodos.

[26] Tabela original do artigo de Sezer & Ozbayoglu (2019)

Por outro lado, no primeiro caso o algoritmo foi capaz de dar uma melhor resposta face às perdas que foram geradas (**RoMaD** de 121,31% vs 61,62%).

O ponto fraco deste algoritmo é evidenciado no valor do *Idle Ratio*, visto que o modelo está parado em quase 50% dos casos. Durante períodos *trendless* ou *bear markets* este fenómeno não é um problema (se estivermos também a falar de momentos de queda acaba por ser uma das melhores estratégias durante este tipo de período). Contudo durante *bull markets*, ficar neste "impasse" leva a custos de oportunidade e perda de potenciais lucros e, em suma, os retornos vão diminuir se não se houver transações de *buy-sell*. Esta será provavelmente uma das razões para que a CNN não seja melhor que a *Buy and Hold* no período entre 2012-2017, visto que o modelo não fez nenhuma transação e ficou *on cash* durante 50% das vezes.

Uma das sugestões dos autores passa por este ponto, pois se se conseguir estratégias que diminuem o *Idle Ratio* os resultados poderão melhorar. Isto pode ser alcançado de diferentes formas, em vez de processar e fazer *trading* de uma única ação, mas sim de um *basket* de ações (não correlacionadas). Isto pode levar ao aumento do número anual de transações, sem comprometer a taxa de sucesso, o que irá resultar num menor *Idle ratio* e um maior retorno anual. De uma forma geral o modelo teria de ser ajustado para ser mais agressivo durante *bull markets*.

Neste capítulo tal como referido acima apenas foi descrito o que foi desenvolvido com os dados originais e as respetivas conclusões. Para o modelo ajustado não foi possível contemplar também esta parte visto que levaria a que o trabalho final ficasse com uma extensão para além do que é pedido e por outro lado seria necessário mais tempo para aprofundar de forma assertiva neste ponto. Devido a este fim ter ficado em aberto, o objetivo agora será o início de um novo projeto de trabalho de investigação relacionado com o tema apresentado nesta dissertação.

Conclusão

Esta dissertação tinha o objetivo de estudar o desenvolvimento de um algoritmo de estratégia de *Trading* "out-of-the-box" que se baseava em identificar pontos de *Buy-Sell* através de um modelo que recorre a uma CNN treinada usando imagens de gráficos de barras do preço fecho de ações.

Esta abordagem foi um desafio visto que todas as metodologias usam dados de séries temporais das ações direta ou indiretamente e como tal a literatura sobre a mesma ainda se encontra em número reduzido quando em comparação com outros temas de *Machine Learning*.

Numa primeira parte tentou perceber-se os temas que o estudo aborda, como funciona uma rede neuronal convolucional e, por outro lado, o que já tinha sido desenvolvido nos Mercados Financeiros e em especial para modelos de *Trading*. Após esse entendimento, o foco foi passar para o desenvolvimento feito por Sezer & Ozbayoglu (2019). Nesse estudo utilizaram os dados de *Dow Jones 30* de 1997 a 2017 (5035 dias) com dois diferentes conjuntos de dados de teste (2007-2012 e 2012-2017), porém ao estudar esta parte percebeu-se que existiria um ponto que talvez pudesse ser melhorado, pois apesar do modelo apresentar uma *accuracy* bastante alta na fase de treino, o mesmo não acontecia quando se passava para a fase de teste. Um dos principais motivos para este problema residia na parte de Rotulagem das imagens, visto que as imagens com Rótulo igual a 1 (Compra) ou 2 (Venda) foram classificadas corretamente na sua maioria e o mesmo não estava a acontecer com as imagens que deveriam ter rótulo igual a 0 (Manter).

Por conseguinte, na segunda parte deste trabalho, após estudar e testar qual a melhor abordagem a usar, que dados utilizar, como rotular as imagens, como melhorar a arquitetura da rede com o objetivo de equilibrar o processo de rotulagem sem ter *overfitting* foi também esta a fase mais desafiante deste trabalho. Estendeu-se a janela temporal dos dados, passando o modelo a contemplar os preços de fecho de *Dow Jones 30* de 1987 a 2020 (8518 dias), sendo os dados de treino desde 1987 a 2010 (6053 dias) e os dados de teste de 2011 até 2020 (2465 dias). O processo que leva à rotulagem das imagens é baseado no declive dos preços que compõe cada imagem. Nesta parte, alterou- se o declive e considerou-se que este fosse calculado entre um número menor de dias futuros. Por fim e devido à complexidade que existe na arquitetura da rede e sendo esta parte uma limitação que senti visto que esta é ainda uma área nova para mim, decidiu-se não alterar a arquitetura da rede e apenas se reduziu o número de *epochs* da CNN.

Considera-se que resultados obtidos na avaliação da rede neuronal são promissores, apesar da *accuracy* não verificar um valor muito elevado, neste primeiro contributo o objetivo era que o algoritmo fosse capaz de classificar de forma correta ou pelo menos mais consistente as 3 classes; por isso o foco centrou-se mais nos resultados da *Confusion matrix* do que no valor da *accuracy* e esse objetivo foi alcançado.

Do ponto de vista de avaliação do modelo de *Trading* os resultados obtidos por Sezer & Ozbayoglu (2019) demonstraram que o modelo proposto foi capaz de gerar lucro de uma forma consistente, tal como obter uma melhor performance em comparação com uma estratégia de *Buy and Hold* dependendo das condições de mercado. Este é um ponto fundamental para a validação do modelo, pois mais do que obter um modelo que alcance uma precisão elevada, o objetivo é que este obtenha lucro. Infelizmente, devido ao tempo que esta parte acarreta de forma a apresentar resultados corretos e concretos não foi

possível que o modelo alterado nesta dissertação passa-se também por esta validação. O desejo é agora continuar a desenvolver este tema através de um novo projeto de investigação sobre o mesmo.

De uma forma geral, e tendo em conta que esta foi uma das primeiras abordagens neste campo, os resultados obtidos e apresentados são promissores como também revelam que são várias as melhorias que podem ser feitas, desde a melhoria da fase de rotulagem, escolha de outros dados, mudanças na arquitetura da rede ou até mesmo a escolha de um modelo híbrido.

Referências Bibliográficas

Bao, Y., Yang, Y., Xiong T., & Zhang,J. (2011).A Comparative Study of Multi-step-ahead Prediction for Crude Oil Price with Support Vector Regression. Fourth International Joint Conference on Computational Sciences and Optimization, Yunnan, 598-602.

Cavalcante, R.C., Brasileiro, R.C., Souza, V., Nobrega, J.P., & Oliveira A.L.I. (2016). Computational intelligence and financial markets: A survey and future directions, Expert Systems with Applications 55, 194-211.

Ding, X. (2015). Deep Learning for Event-Driven Stock Prediction, International Joint Conference on Artificial Intelligence, 2327-2333.

Fama, E. (1965). The Behaviour of Stock Market Prices. Journal of Business, 64, 34-105.

González, A. R., García-Crespo, Á., Palacios, R. C., Guldrís-Iglsesias, F., & Berbís, J. M. G. (2011). CAST: Using neural networks to improve trading systems based on technicalanalysis by means of the RSI financial indicator. *Expert Systems With Applications*, 38(9), 11489-11500.

Goodfellow, Y. B., & Courville, A. (2016) .Deep Learning, MIT Press.

Hinton, G. E., Lang, K. J., & Waibel A. H.(1989). A Time-Delay Neural Network Architecture for Isolated Word Recognition, 3, 23-43.

Hinton, G., Krizhevsky, A., & Sutskever, I., (2012). ImageNet classification with deep convolutional neural networks, Advances in neural information processing systems, 1097-1105.Keras: The Python Deep Learning library, "Keras Documentation. Disponível em: https://keras.io/. (Acedido a: 20-03-2020)

Hubel, D. H. & Wiesel, T. N. (1989). "Receptive fields and functional architecture of monkey striate cortex," Journal of Physiology (London), vol. 195, pp. 215–243.

Kingman,D. J., & Ba, J. L. (2015).Adam: A Method For Stochastic Optimization.

Krizhevsky, A., Sutskever, I. & Hinton,G. (2012). ImageNet classification with deep convolutional neural networks, Advances in neural information processing systems, pp. 1097-1105.

LeCun, Y., Boser, B., Denker, J. S. , Henderson, D. , Howard, R. E. , Hubbard, W., & Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition.Neural Computation 1:4, 541-551,

LeCun, Y., Bengio Y. & Hinton, G. (2015). Deep learning, Nature 521.7553, 436-444.

Mack, D. (2018). How to pick the best learning rate for your machine learning project. https://medium.com/octavian-ai/which-optimizer-and-learning-rate-should-i-use-for-deep-learning-5acb418f9b2 (Acedido a: 03-05-2020)

Matthew, D. Z. (2012). ADADELTA: An Adaptive Learning Rate Method.

Nayak, A. (2019). Stock Buy/Sell Prediction Using Convolutional Neural Network. https://towardsdatascience.com/stock-market-action-prediction-with-convnet-8689238feae3(Acedido a 04-05-2020)

Rumelhart, D., Hinton, G. & Williams, R. Learning representations by back-propagating errors. Nature 323, 533–536 (1986).

Rocca, B., & Rocca, J. (2019). Handling imbalanced datasets in machine learning. https://towardsdatascience.com/handling-imbalanced-datasets-in-machine-learning-7a0e84220f28 (Acedido 03-05-2020)

Sezer, O.B., Ozbayoglu, A.M., & Dogdu E. (2017).An Artificial Neural Network-based Stock Trading System Using Technical Analysis and Big Data Framework, ACM Proceedings of the South East Conference, 223-226.

Sezer, O.B., Ozbayoglu, A.M., & Dogdu E. (2017). A Deep Neural-Network based Stock Trading System based on Evolutionary Optimized Technical Analysis Parameters, Complex Adaptive Systems Conference, 114, 473–480.

Sezer, O.B., & Ozbayoglu, A.M. (2018). Algorithmic Financial Trading with Deep Convolutional Neural Networks: Time Series to Image Conversion Approach Applied Soft Computing, 70, 525–538.

Sezer, O.B. & Ozbayoglu, A.M. (2019).Financial Trading Model with Stock Bar Chart Image Time Series with Deep Convolutional Neural Networks.

Soni, S. (2011). Applications of ANNs in Stock Market Prediction: A Survey/ Journal of Computer Science & Engineering Technology (IJCSET) ISSN: 2229-3345 Vol. 2 No. 3 71 2011

Volpi. F. G. (2019). Class Imbalance: a classification headache. https://towardsdatascience.com/class-imbalance-a-classification-headache-1939297ff4a4. (Acedido a 03-05-2020)

Zhang, G., Patuwo B. E., & Hu, M. Y. (1998). Forecasting with artificial neural networks: The state of the art, International Journal of Forecasting, 14, 35-62.

Zhou, Y.T., & Chellappa, R. (1988). Computation of optical flow using a neural network. IEEE 1988 International Conference on Neural Networks, 71-78 vol.2.

Anexos

Anexo A

O Anexo A é o código de programação em ***Python***, cujo objetivo foi o tratamento dos dados de forma a calcular os declives, formar as imagens e fazer a sua legenda para depois serem utilizados para o treino e teste da rede neuronal. Este processo que resultou no código apresentado foi explicado em pormenor no ponto **3.1 Método**. O programa neste anexo foi retirado de Sezer & Ozbayoglu (2019) e foi alterado.

```
import numpy as np

import pandas as pd

import csv

import matplotlib.pyplot as pl

import matplotlib.image as mpimg



training_input_file="data/MSFT19872010.csv"

test_input_file="data/MSFT20112020.csv"

training_output_file="training.csv"

test_output_file="test.csv"


# calculate slopeReference (slopeRef) and find the distribution of the labels

# find the first,second seperation points..

def get_histogram(input_name,roc):

    df = pd.read_csv(input_name, header=None, index_col=None, delimiter=',')

    for r in range(len(df) - 34):

        all_y = df[5].values.tolist()

        price45 = all_y[r + 33]

        #print(price45)

        price30 = all_y[r + 29]
```

```
        #print(price30)
        dif_ratio = ((price45 - price30) / price30) * 100
        roc.append(dif_ratio)
    roc.sort()
    len_roc=len(roc)
    f_sliding=roc.__getitem__(2*int(len_roc/5))
    s_sliding = roc.__getitem__(3*int(len_roc/5))
    print("len_roc:",len_roc, "f_sliding:",f_sliding,"s_sliding:",s_sliding)
    #print(len(df))
    #print(roc)
    # my_hy = pd.DataFrame(all_y)
    # my_hr = pd.DataFrame(roc)
    # imgploty = my_hy.pl.hist()
    # imgplotr = my_hr.pl.hist()
    # pl.show()
    return round(f_sliding,2),round(s_sliding,2)



# imagesFile
def imagesFileCreation(input_name,output_name,f_sliding,s_sliding):
    # read csv file
    df = pd.read_csv(input_name, header=None, index_col=None, delimiter=',')
    # for all value, create image file
    for r in range(len(df) - 45):
        #print('r:', r)
        img = [[]]
```

```
# all pixels are set value 255 (white) in 30x30 pixel image
img = [[255 for i in range(30)] for i in range(30)]
#print (img)
all_y = df[5].values.tolist()
sub_y = all_y[r:r + 30]

current_price = round(all_y[r+29], 2)

price45 = all_y[r + 33]
price30 = all_y[r + 29]

# calculate ratio
dif_ratio = ((price45 - price30) / price30) * 100



#
max_y = (all_y[r+15] * 130) / 100
min_y = (all_y[r+15] * 70) / 100

label_avg_y = 0
label_sum_y = 0

print("f_sliding,s_sliding:",f_sliding,s_sliding)
#test1
if (dif_ratio >= s_sliding): # slopeCurrent>slopeRef[s_sliding]
    predictLabel = 1   # label=Buy
```

```
elif (dif_ratio > f_sliding and dif_ratio < s_sliding):
    predictLabel = 0   # label = Hold
elif (dif_ratio <= f_sliding): # slopeCurrent<slopeRef[s_sliding]
    predictLabel = 2   # label = Sell

print("predictLabel:", predictLabel, " dif_ratio:", dif_ratio, "price:", current_price)
print("max:",max_y, "min:",min_y)

# calculate coefficient to normalize data
coef = 30 / (int(max_y - min_y)+1)
#print(coef)
j = 0
#print(max_y, min_y)

# calculate the stock price and create black bar graphics for 30 days
for i in range(30):
    val = (sub_y[i] - min_y) * coef
    #print(val)
    for k in range(int(val)):
        if(k<30):
            img[29 - k][j] = 0
    j += 1
    #print (img)
#print(j)

my_df = pd.DataFrame(img)
```

```
        label_price = ';'.join((str(predictLabel), str(current_price)))

        # append image values in file
        my_df.to_csv(output_name, index=False, header=False, mode='a', line_terminator=';', sep=';')
        with open(output_name, 'a') as file:
            file.write(label_price)
            file.write('\n')
        del label_sum_y, predictLabel

        # print image

        #imgplot = pl.imshow(img, cmap=pl.get_cmap('gray'))
       # pl.show()

if __name__=="__main__":
    roc = []
    f_sliding,s_sliding=get_histogram(training_input_file,roc)
    print("Starting Training File Generation..")
    imagesFileCreation(training_input_file, training_output_file,f_sliding,s_sliding)
    print("Ending Training File Generation...")
    print("Starting Test File Generation..")
    imagesFileCreation(test_input_file, test_output_file,f_sliding,s_sliding)
    print("Ending Test File Generation...")
```

Anexo B

O Anexo B é o código de programação em ***Python***. Neste ficheiro está programada a rede neuronal utilizada, tal como a avaliação do ponto de vista da precisão do modelo. A arquitetura da rede que resultou neste código foi explicada no ponto **3.1 Método** e as métricas para esta avaliação foram referidas em **3.2 Avaliação dos Resultados**, A) Avaliação da Rede Neuronal. O programa neste anexo foi retirado de Sezer & Ozbayoglu (2019) e foi alterado.

```
import numpy as np

import pandas as pd

import csv

from sklearn.model_selection import StratifiedKFold

from sklearn import metrics

import matplotlib.pyplot as plt

import keras

import tensorflow

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten

from tensorflow.keras.layers import Conv2D, MaxPooling2D

from tensorflow.keras import regularizers

from tensorflow.keras.regularizers import l2


from sklearn.metrics import classification_report, confusion_matrix

from sklearn.utils import shuffle

import math
```

```
def train_cnn(training_df, test_df, params):
    # Trains and evaluates CNN on the given train and test data, respectively.

    print("Training is starting ...")
    #quero t-odas as linhas e as colunas até 899, ou seja, todas menos as 2 ultimas
    #train_images = training_df.ix[:, :899].as_matrix()
    print(training_df.iloc[:, :900].values)
    # train_labels = training_df.ix[:, 900]
    # train_prices = training_df.ix[: ,901]

    # test_images = test_df.ix[:, :899].as_matrix()
    # test_labels = test_df.ix[:, 900]
    # test_prices = test_df.ix[:, 901]
    train_images = training_df.iloc[:, :900].values
    train_labels = training_df.iloc[:, 900]
    train_prices = training_df.iloc[:, 901]

    test_images = test_df.iloc[:, :900].values
    print(type(test_images))
    print(type(params["batch_size"]))
    test_labels = test_df.iloc[:, 900]
    test_prices = test_df.iloc[:, 901]
```

```
    test_labels = keras.utils.to_categorical(test_labels, params["num_classes"])
    train_labels = keras.utils.to_categorical(train_labels, params["num_classes"])


    print(train_images.shape)
    print(train_labels)
    print(test_prices)
    train_images = train_images.reshape(train_images.shape[0], params["input_w"], params["input_h"],
1)
    test_images = test_images.reshape(test_images.shape[0], params["input_w"], params["input_h"], 1)




    # CNN model
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(params["input_w"],
params["input_h"], 1)))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(128,activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(params["num_classes"],activation='softmax'))
    model.compile(loss= tensorflow.keras.losses.categorical_crossentropy,
                optimizer=tensorflow.keras.optimizers.Adadelta(),
                metrics=['accuracy', 'mae', 'mse'])
```

```
    # metrics.accuracy_score, metrics.recall_score, metrics.average_precision_score,
metrics.confusion_matrix

    train_data_size = train_images.shape[0]

    test_data_size = test_images.shape[0]




    print("model will be trained with {} and be tested with {}
sample".format(train_data_size,test_data_size))

    # fit the model to the training data

    print("Fitting model to the training data...")

    print("")

    history=model.fit(train_images, train_labels,validation_data=(test_images, test_labels),
batch_size=params["batch_size"], epochs=params["epochs"], verbose=1)


    predictions = model.predict(test_images, batch_size=params["batch_size"], verbose=1)

    print("loss, accuracy, mean absolute error, mean squared error")

    print(model.evaluate(test_images, test_labels, batch_size=params["batch_size"], verbose=1))


    print("Train conf matrix: ", confusion_matrix(np.array(reverse_one_hot(train_labels)),

                                    np.array(reverse_one_hot(model.predict(train_images,
batch_size=params["batch_size"], verbose=1)))))


    print("Test conf matrix: ",  confusion_matrix(np.array(reverse_one_hot(test_labels)),

                                    np.array(reverse_one_hot(predictions))))

    print("Train classification:",classification_report(np.array(reverse_one_hot(train_labels)),

                                    np.array(reverse_one_hot(model.predict(train_images,
batch_size=params["batch_size"], verbose=1)))))
```

```
    print("Test classification:", classification_report(np.array(reverse_one_hot(test_labels)),
                                                       np.array(reverse_one_hot(predictions))))
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title('model accuracy')
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()
    # summarize history for loss
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()


    return predictions, test_labels, test_prices



# reverse one hot manually
def reverse_one_hot(predictions):
```

```
    reversed_x = []
    for x in predictions:
        reversed_x.append(np.argmax(np.array(x)))
    return reversed_x




# read csv file and create df files
train_df = pd.read_csv("training.csv", header=None, index_col=None, delimiter=';')
test_df = pd.read_csv("test.csv", header=None, index_col=None, delimiter=';')



print("Before data imbalance")
#cria matriz com 902 colunas e linhas que tem na coluna 900 o o valor zero. sao 676 linhas
l0_train = train_df.loc[train_df[900] == 0]
l1_train = train_df.loc[train_df[900] == 1]
l2_train = train_df.loc[train_df[900] == 2]
#shape 0 da-me o numero de linhas de cada matriz de treino
l0_train_size = l0_train.shape[0]
l1_train_size = l1_train.shape[0]
l2_train_size = l2_train.shape[0]
print('train_df => l0_size:',l0_train_size,'l1_size:',l1_train_size,'l2_size:',l2_train_size)
l0_test = test_df.loc[test_df[900] == 0]
l1_test = test_df.loc[test_df[900] == 1]
l2_test = test_df.loc[test_df[900] == 2]
```

```
l0_test_size = l0_test.shape[0]

l1_test_size = l1_test.shape[0]

l2_test_size = l2_test.shape[0]

print('test_df => l0_size:',l0_test_size,'l1_size:',l1_test_size,'l2_size:',l2_test_size)


# calculate the number of labels and find ratios

l0_l1_ratio = (l0_train_size//l1_train_size)

l0_l2_ratio = (l0_train_size//l2_train_size)

l1_l0_ratio = (l1_train_size//l0_train_size)

l1_l2_ratio = (l1_train_size//l2_train_size)

l2_l0_ratio = (l2_train_size//l0_train_size)

l2_l1_ratio = (l2_train_size//l1_train_size)

print("l0_l1_ratio:",l0_l1_ratio)

print("l0_l2_ratio:",l0_l2_ratio)

print("l1_l0_ratio:",l1_l0_ratio)

print("l1_l2_ratio:",l1_l2_ratio)

print("l2_l0_ratio:",l2_l0_ratio)

print("l2_l1_ratio:",l2_l1_ratio)

#if there is data imbalance, solution of data imbalance in training set

#data imbalance #0/#1>1


# to solve data imbalance

# l0_new = pd.DataFrame()

# l1_new = pd.DataFrame()

# l2_new = pd.DataFrame()

# for idx, row in train_df.iterrows():
```

```
#     if row[900] == 0:
#         for i in range(2):
#             l0_new = l0_new.append(row)
#     if row[900] == 1:
#         for i in range(2):
#             l1_new = l1_new.append(row)
#     if row[900] == 2:
#         for i in range(2):
#             l2_new = l2_new.append(row)

# train_df = train_df.append(l0_new)
# train_df = train_df.append(l1_new)
# train_df = train_df.append(l2_new)
#
# shuffle
train_df = shuffle(train_df)



print("After data imbalance")
l0_train = train_df.loc[train_df[900] == 0]
l1_train = train_df.loc[train_df[900] == 1]
l2_train = train_df.loc[train_df[900] == 2]
l0_train_size = l0_train.shape[0]
l1_train_size = l1_train.shape[0]
l2_train_size = l2_train.shape[0]
print('train_df => l0_size:',l0_train_size,'l1_size:',l1_train_size,'l2_size:',l2_train_size)
```

```
l0_test = test_df.loc[test_df[900] == 0]

l1_test = test_df.loc[test_df[900] == 1]

l2_test = test_df.loc[test_df[900] == 2]

l0_test_size = l0_test.shape[0]

l1_test_size = l1_test.shape[0]

l2_test_size = l2_test.shape[0]

print('test_df => l0_size:',l0_test_size,'l1_size:',l1_test_size,'l2_size:',l2_test_size)

train_df.reset_index(drop=True, inplace=True)

test_df.reset_index(drop=True, inplace=True)


print("train_df size: ", train_df.shape)


# parameters of the cnn

params = {"input_w": 30, "input_h": 30, "num_classes": 3, "batch_size": 1024, "epochs": 50}


#call cnn

predictions, test_labels, test_prices = train_cnn(train_df, test_df, params)
```

Anexo C

O anexo C é o código de programação em ***Java***, cujo objetivo foi a avaliação do modelo na perspectiva de *trading*, utilizando as métricas referidas **3.2 Avaliação dos Resultados**, B) Avaliação do modelo de *trading*. O programa neste Anexo C foi retirado de Seze & Ozbayoglu (2019).

```
package ml.cnn.image.paper;



import java.io.BufferedWriter;

import java.io.DataInputStream;

import java.io.FileInputStream;

import java.io.FileWriter;

import java.io.IOException;

import java.util.regex.Matcher;

import java.util.regex.Pattern;


import org.apache.commons.math3.util.Precision;


public class evaluateFinancial {



	public static void main(String[] args) throws Exception  {

		StringBuilder builder = new StringBuilder();

		String fName = "resources4/cnn_result.csv";

		String thisLine;

		int count=0;

		FileInputStream fis = new FileInputStream(fName);
```

```
DataInputStream myInput = new DataInputStream(fis);

int i=0, totalTransactionLength=0;

Double buyPoint,sellPoint,gain=0.0,totalGain=0.0, money=10000.0,
shareNumber=0.0, moneyTemp=0.0, maximumMoney=0.0,
minimumMoney=10000.0,maximumGain=0.0, maximumLost=100.0, totalPercentProfit=0.0;

int transactionCount=0, successTransactionCount=0,failedTransactionCount=0;

Double buyPointBAH,shareNumberBAH,moneyBAH=10000.0,
maximumProfitPercent=0.0, maximumLostPercent=0.0;

Boolean forceSell=false;




String[][] data = new String[0][];//csv data line count=0 initially

while ((thisLine = myInput.readLine()) != null) {

    ++i;//increment the line count when new line found


    String[][] newdata = new String[i][2];//create new array for data


    String strar[] = thisLine.split(";");//get contents of line as an array

    newdata[i - 1] = strar;//add new line to the array


    System.arraycopy(data, 0, newdata, 0, i - 1);//copy previously read values to
new array

    data = newdata;//set new array as csv data

}
```

```
/*System.out.println("------------------------------"); //debug print data array
for (String[] strings : data) {
	for (String string : strings) {
		System.out.print("\t" + string);
	}
	System.out.println();
}*/


int k=0;
System.out.println("Start Capital: \\$" + money);
builder.append("Start Capital: \\$" + money+"\n");

while(k<data.length-1){

	if(Double.valueOf(data[k][0])==1.0){

		buyPoint=Double.valueOf(data[k][2]);
		buyPoint=buyPoint*100;
		shareNumber=(money-1.0)/buyPoint;
		forceSell=false;
		for (int j=k; j<data.length-1; j++) {

			sellPoint=Double.valueOf(data[j][2]);
			sellPoint=sellPoint*100;
			moneyTemp=(shareNumber*sellPoint)-1.0;
```

```
//stop loss %10
/*if(money*0.9>moneyTemp){
	money=moneyTemp;
	forceSell=true;
}*/
//if(Double.valueOf(data[j][0])==0.0 || forceSell==true){
if(Double.valueOf(data[j][0])==2.0 || forceSell==true){
	sellPoint=Double.valueOf(data[j][2]);
	sellPoint=sellPoint*100;
	gain=sellPoint-buyPoint;
	if(gain>0){
		successTransactionCount++;
	}
	else{
		failedTransactionCount++;
	}

	if(gain>=maximumGain){
		maximumGain=gain;

maximumProfitPercent=Double.valueOf(maximumGain)/Double.valueOf(buyPoint)*100;

	}
	if(gain<=maximumLost){
		maximumLost=gain;

maximumLostPercent=Double.valueOf(maximumLost)/Double.valueOf(buyPoint)*100;
```

```
                                        }
                                        moneyTemp=(shareNumber*sellPoint)-1.0;
                                        money=moneyTemp;
                                        if(money>maximumMoney){
                                                maximumMoney=money;
                                        }
                                        if(money<minimumMoney){
                                                minimumMoney=money;
                                        }
                                        transactionCount++;

        //System.out.println("\\\\"+transactionCount+"."+"("+(k+1)+"-"+(j+1)+") => " +
Precision.round(sellPoint,2) + "-" + Precision.round(buyPoint,2)+ "= " + Precision.round(gain,2) + "
Capital: \\$" + Precision.round(money,2) );

        System.out.println(transactionCount+"."+"("+(k+1)+"-"+(j+1)+") => " +
Precision.round((gain*shareNumber),2) + " Capital: $" + Precision.round(money,2) );
                                        builder.append(transactionCount+"."+"("+(k+1)+"-
"+(j+1)+") => " + Precision.round((gain*shareNumber),2) + " Capital: $" +
Precision.round(money,2)+"\n");

                                        //System.out.println(Precision.round(money,2) );
                                        totalPercentProfit=totalPercentProfit
+(gain/buyPoint);

                                        totalTransactionLength=totalTransactionLength+(j-
k);
                                        k=j+1;
                                        totalGain=totalGain+gain;
                                        break;
```

```
                        }
                    }
                }
                k++;
            }

            System.out.println("Our System => totalMoney = $" + Precision.round(money,2) );
            builder.append("Our System => totalMoney = $" + Precision.round(money,2)+"\n");

            buyPointBAH=Double.valueOf(data[0][2]);
            shareNumberBAH=(moneyBAH-1.0)/buyPointBAH;
            moneyBAH=(Double.valueOf(data[data.length-1][2])*shareNumberBAH)-1.0;


            System.out.println("BAH => totalMoney = $" + Precision.round(moneyBAH,2) );
            builder.append("BAH => totalMoney = $" + Precision.round(moneyBAH,2)+"\n");

            double numberOfDays=Double.valueOf(data.length-1);
            double numberOfYears=numberOfDays/365;
            //money/10000.0;

            //report the results
            System.out.println("Our System Annualized return % => " +
Precision.round(((Math.exp(Math.log(money/10000.0)/numberOfYears)-1)*100),2) + "%" );
            System.out.println("BaH Annualized return % => " +
Precision.round(((Math.exp(Math.log(moneyBAH/10000.0)/numberOfYears)-1)*100),2) +"%");
            System.out.println("Annualized number of transaction => " +
Precision.round((Double.valueOf(transactionCount)/numberOfYears),1) + "#");
```

```
System.out.println("Percent success of transaction => "
+Precision.round((Double.valueOf(successTransactionCount)/Double.valueOf(transactionCount))*10
0,2)+ "%");

System.out.println("Average percent profit per transaction => " +
Precision.round((totalPercentProfit/transactionCount*100),2) + "%" );

System.out.println("Average transaction length => " +
totalTransactionLength/transactionCount + "#");

System.out.println("Maximum profit percent in transaction=> " +
Precision.round(maximumProfitPercent,2) + "%");

System.out.println("Maximum loss percent in transaction=> " +
Precision.round(maximumLostPercent,2)+ "%");

System.out.println("Maximum capital value=> " + "$"+
Precision.round(maximumMoney,2));

System.out.println("Minimum capital value=> " + "$"+
Precision.round(minimumMoney,2));

System.out.println("Idle Ratio %=>  " +
Precision.round((Double.valueOf(data.length-
totalTransactionLength)/Double.valueOf(data.length)*100),2) + "%");
```

```
builder.append("Our System Annualized return % => " +
Precision.round(((Math.exp(Math.log(money/10000.0)/numberOfYears)-1)*100),2) + "%" +"\n");

builder.append("BaH Annualized return % => " +
Precision.round(((Math.exp(Math.log(moneyBAH/10000.0)/numberOfYears)-1)*100),2) +"%"+"\n");

builder.append("Annualized number of transaction => " +
Precision.round((Double.valueOf(transactionCount)/numberOfYears),1) + "#"+"\n");

builder.append("Percent success of transaction => "
+Precision.round((Double.valueOf(successTransactionCount)/Double.valueOf(transactionCount))*10
0,2)+ "%"+"\n");
```

```
		builder.append("Average percent profit per transaction => " +
Precision.round((totalPercentProfit/transactionCount*100),2) + "%" +"\n");

		builder.append("Average transaction length => " +
totalTransactionLength/transactionCount + "#"+"\n");

		builder.append("Maximum profit percent in transaction=> " +
Precision.round(maximumProfitPercent,2) + "%"+"\n");

		builder.append("Maximum loss percent in transaction=> " +
Precision.round(maximumLostPercent,2)+ "%"+"\n");

		builder.append("Maximum capital value=> " + "$"+
Precision.round(maximumMoney,2)+"\n");

		builder.append("Minimum capital value=> " + "$"+
Precision.round(minimumMoney,2)+"\n");

		builder.append("Idle Ratio %=>  " +  Precision.round((Double.valueOf(data.length-
totalTransactionLength)/Double.valueOf(data.length)*100),2) + "%"+"\n");


		BufferedWriter writer;

		try {

			writer = new BufferedWriter(new FileWriter("resources4/Results.txt"));

			writer.write(builder.toString());//save the string representation of the board

			writer.close();

		} catch (IOException e) {

			// TODO Auto-generated catch block

			e.printStackTrace();

		}



	}


}
```